ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО

Приглашенный преподаватель
департамента программной инженерии
факультета компьютерных наук, бакалавр

_____ Г. М. Сосновский
«_____» _____ 2024 г.

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия»

_____ Н. А. Павлочев
«_____» _____ 2024 г.

ОБРАЗОВАТЕЛЬНАЯ ПЛАТФОРМА. ПОДСИСТЕМА «COURSES»

Пояснительная записка

Лист Утверждения

RU.17701729.09.12-01 81 01-1-ЛУ

Исполнитель: Студент группы БПИ 217
_____ Киселёв И. А.
«_____» _____ 2024 г.

Москва 2024

Утверждён
RU.17701729.09.12-01 81 01-1-ЛУ

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

ОБРАЗОВАТЕЛЬНАЯ ПЛАТФОРМА. ПОДСИСТЕМА «COURSES»

Пояснительная записка

RU.17701729.09.12-01 81 01-1

Листов 25

Москва 2024

Аннотация

Данная Пояснительная записка к проекту «Образовательная платформа» содержит следующие разделы: «Введение», «Назначение и область применения», «Технические характеристики», «Ожидаемы технико-экономические показатели», «Источники, использованные при разработке».

В разделе «Введение» дается наименование программы на русском и английском языках и документ, на основании которого ведется разработка.

В разделе «Назначение и область применения» описана область применения разработки.

В разделе «Технические характеристики» описаны постановка задачи, применяемы математические методы, алгоритмы и функционирование программы, описание и обоснование выбора метода организации входных и выходных данных, описание и обоснование выбора состава технических средств, описание и обоснование выбора состава программных средств.

В разделе «Ожидаемые технико-экономические показатели» описано сопоставление с аналогами разрабатываемой системы.

Раздел «Источники, использованные при разработке» представлены источники информации.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению;
2) ГОСТ 19.101-77. Виды программ и программных документов
3) ГОСТ 19.103-77. Обозначение программ и программных документов
4) ГОСТ 19.104-78 Основные надписи;
5) ГОСТ 19.105-78 Общие требования к программным документам;
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом [8];
7) ГОСТ 19.404-79 Пояснительная записка.[11]

Изменения к данной Пояснительной записке оформляются согласно:

1) ГОСТ 19.603-78.[8]
2) ГОСТ 19.604-78. [9]

Содержание

1. Введение

1.1. Наименование программы

1.1.1. Наименование программы на русском языке

Образовательная платформа.

1.1.2. Наименование программы на английском языке

Educational platform.

1.2. Краткая характеристика области применения

Система предназначена для пользования людьми, желающими изучать и проходить курсы в интернете, а также пользоваться дополнительным функционалом при изучении, например, статистики прохождения курсов, уведомления, доступ в чаты, писать отзывы на курсы.

Учителя смогут создавать и размещать свои курсы на данной платформе, получая прибыль с их продажи.

Подсистема Courses используется и интегрируется в общую систему и отвечает за работу с информацией курсов и их предметной области.

1.3. Документы, на основании которых ведется разработка

«Правила подготовки, защиты, оценивания и публикации курсовых проектов студентов ОП ФКН ПИ» Национального исследовательского университета «Высшая школа экономики» от 06.03.2021

2. Назначение и область применения

2.1. Функциональное назначение

Система должна предоставить клиентам возможность просматривать курс, приобретать доступ к курсам и их частям по отдельности, изучать купленные курсы, прослеживать свой прогресс.

Клиент, получив доступ, сможет размещать курсы и получит роль "учитель". Данный пользователь, учитель, сможет создавать и редактировать свои шаблоны курсов, публиковать их.

Пользователи с ролями модератора смогут просматривать курсы (и их содержимое) и обрабатывать заявки на публикацию курсов.

Подсистема отвечает за бизнес логику работы с курсами: доступы для учителей к созданным курсам, учеников к приобретенным, создание и изменение курсов. И не отвечает за авторизацию, аутентификацию, рассылку нотификаций и продажу продуктов (обработку денежных транзакций). Для лучшего понимания назначения подсистемы ее диаграмма прецендентов представлена в Приложении 4 и модель предметной области в Приложении 1.

2.2. Эксплуатационное назначение

Общая система будет представлено в виде нескольких подсистем. Часть подсистем будет реализовать клиентскую часть, веб-браузер с программой, обрабатывающий действия пользователя на веб-сайте. Клиентская часть будет выводить визуализировать необходимые данные пользователю и предоставлять пользовательский интерфейс для работы с информационной системой. Клиентская часть будет получать данные из серверной части.

Серверная часть представляет собой подсистемы, запущенные во внутренней сети. Данные подсистемы будут запущенны в контейнерах на одном или нескольких серверах. Данные подсистемы будут принимать запросы по сети от друг друга или от клиентской части.

Подсистема «Courses» — одна из подсистем серверной части. Подсистему будут ипользовать другие подсистемы серверной части. Взаимодействия подсистем будут происходить посредством HTTP. Схема подсистем и их взаимодействия с подсистемой Courses присутствует в Приложении 2 и Приложении 3.

3. Технические характеристики

3.1. Постановка задачи на разработку программы

Главная задача на подсистему – разработка программы, автоматизирующую процессы создания (структурирования), публикации и модерации курсов, а также выдачи доступов к материалам и частям курсов, и непосредственно выдаче самих курсов и материалов. Программа должна легко интегрироваться в общую систему (использоваться другими подсистемами), а также должна быть расширяемой и масштабируемой. Подсистема при выдаче курсов обычному пользователю (не учителю) должна иметь высокую скорость ответа и высокое целевое значение доступности. Подсистема также должна быть обустроена системами наблюдаемости.

3.1.1. Заинтересованные лица

Лица и их заинтересованность в подсистеме:

- Разработчик:
    1) Простота поддержки и введения новой функциональности
    2) Простота изучения и понимания строения проекта
    3) Простота нахождения ошибок
    4) Простота написания тестов
- Заказчик:
    1) Скорость разработки
    2) Производительность программы
- Учитель:
    1) Простота добавления своих курсов в систему
- Ученик:
    1) Простота изучения курсов и процесса покупки
- Администратор:
    1) Возможность отслеживания статуса и нагрузки подсистемы
- Незарегистрированный в общей системе пользователь:
    1) Возможность получения информации о курсах и ценах

3.1.2. Описание методов проектирования программ

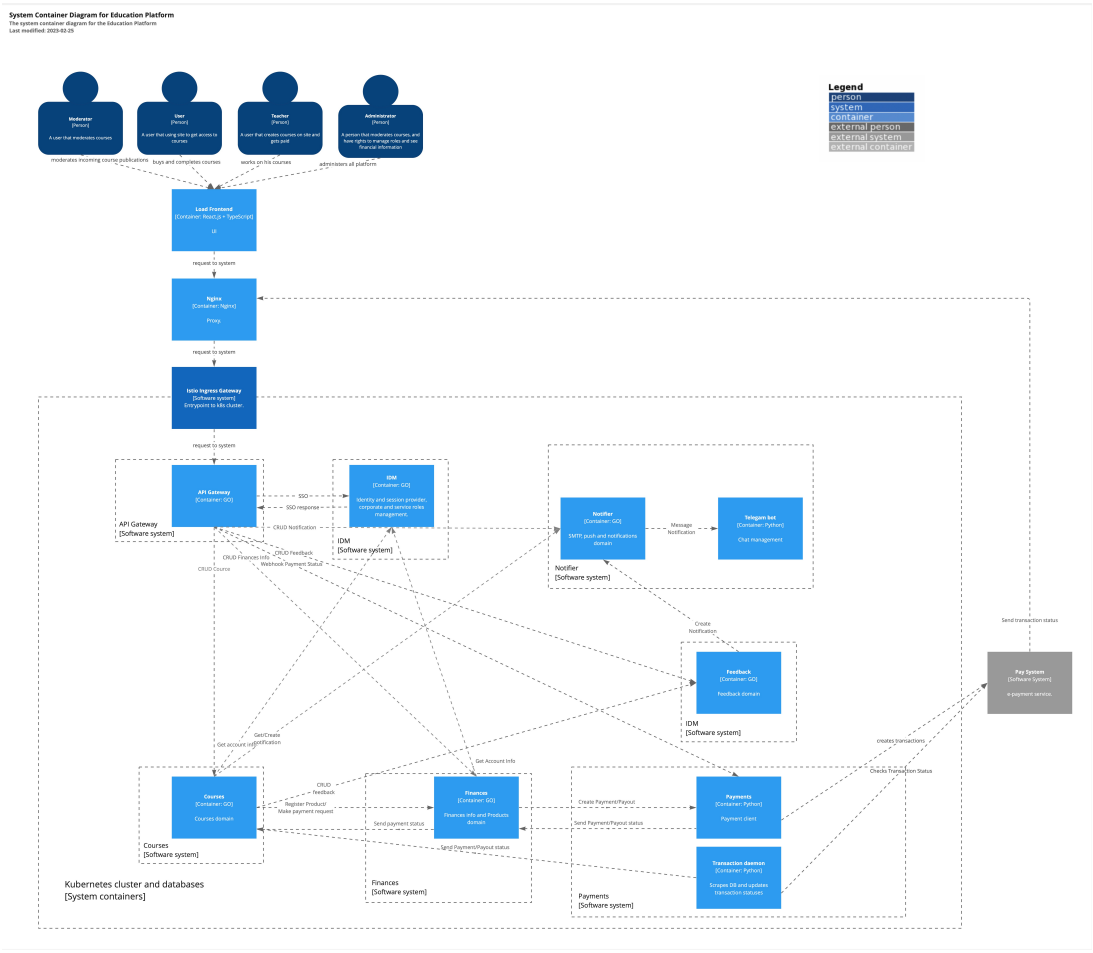
Для поддержании расширяемости и общей системы «Образовательная платформа» была выбрана микросервисная архитектура и информационная система была поделена на микросервисы согласно принципам Domain-Driver-Design[24] по предметным областям. Схема предметной области данной подсистемы после разделения представлена в Приложении 1.

Данная архитектура позволит пере использовать подсистемы платежей («Payments» и «Finances») и уведомлений («Notifications») с других проектах, а также при добавлении функционала, не входящего в предметную область курсов, что требуется заказчику. Согласно принципам данной архитектуры реализация микросервисов не будут зависеть друг от друга, что упростит разработку и обеспечит гибкость (простоту внесения новых требований).

Данная подсистема «Courses» образует собой микросервис, поэтому не должна зависеть от реализации других сервисов, за некоторым исключениями образованными (обоснованными) зависимостями между предметными областями разных систем (см. Приложение 1. Модель предметной области подсистемы), например, курс может хранить в себе идентификатор продукта из «Finances».

Взаимодействие программ представлено на модели уровня контейнеров методологии C4 Model (см. рис. 1)

Рис. 1 — Container level diagram. Edu Platform c4 model



3.1.3. Описание методологии разработки

После долгой итерации разработки с созданием платежной системы и системы уведомлений, а также начального вида системы курсов, заказчик намерен вести разработку короткими итерациями согласно методологии Agile. Разработка текущей системы будет обоснована большим количеством будущих изменений, включая смены основного продукта (разработки другой платформы).

3.2. Описание и обоснование алгоритма функционирования программы

3.2.1. Описание общего функционирования программы

Программа запускается с помощью Docker compose, предварительно используя собранные Docker образы. Docker образы собираются по через Dockerfile, сборка использует multistage build принцип, чтобы в Docker образе не было лишних данных. При своем запуске программа читает файлы конфигураций.

Данная подсистема, принимая по HTTP определённый запрос, с помощью роутера (контролера, который реализует паттерн стратегии), вызывает соответствующий ему метод-хэндлер (handler) класса Handler. Далее данный метод первично обрабатывает информацию из запроса, например проверяя типы данных и поля присланного объекта, и вызывает нужный метод бизнес-логики из реализации интерфейса Service. В методе класса, реализующем сервис происходит основная бизнес-логика прецендента (т.е usecase), данный метод берет данные и объекты из реализации интерфейса Repository (паттерн репозитория). Вызовы методов данных классов, а также сами классы (построены по принципам Чистой архитектуры, подробнее в разделе «Описание архитектуры программы») образуют три слоя — handler, service, repository.

Между тремя данными слоями передаются domain классы (в параметрах и на выходе методов). Однако handler и repository (в отличие от service) не работают с domain классами, а конвертируют их в нужные классы для соотвествующей реализации транспортного протокола или источника данных.

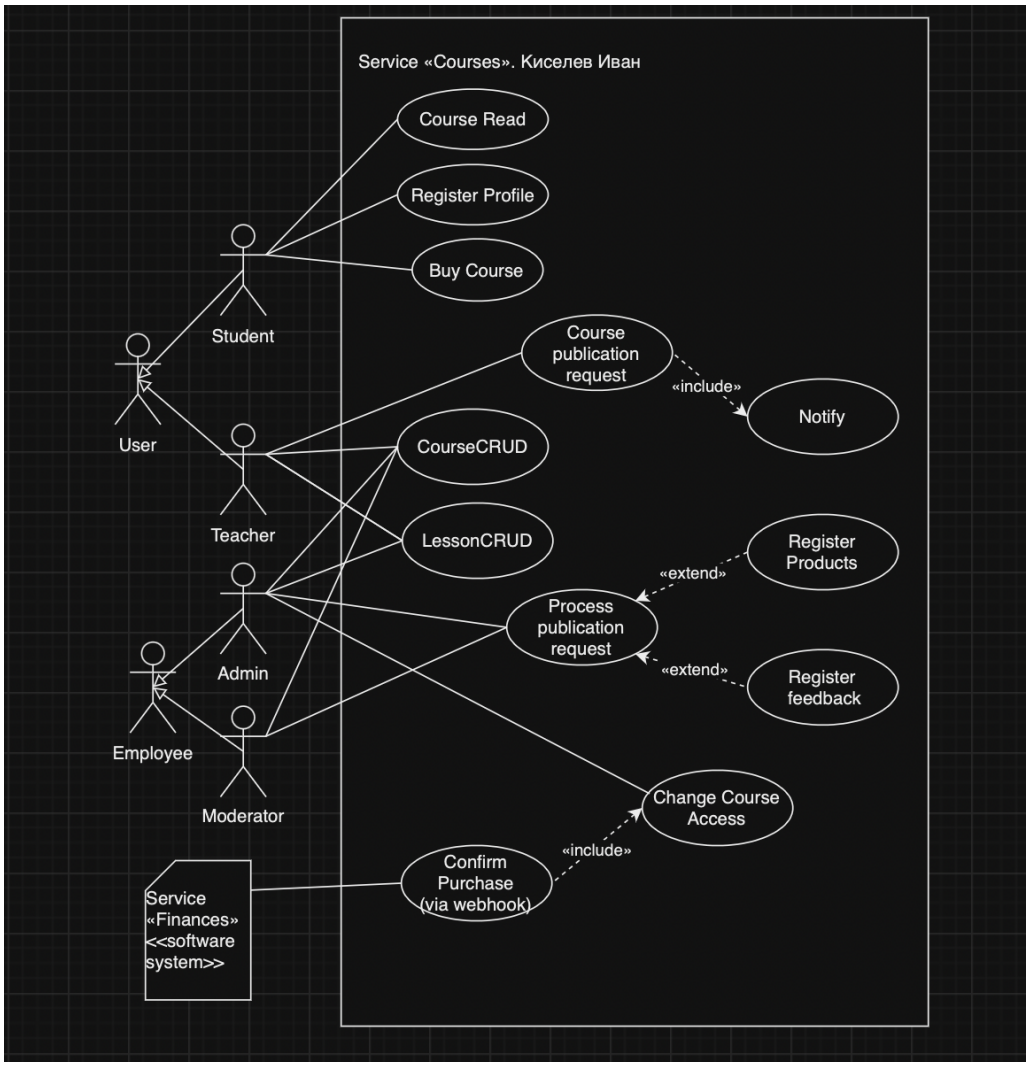
Между данными слоями также передается информация для логгирования и создания трассировки и метрик. Данная информация передается в контексте (Context).

Такая реализация удовлетворяет всем интересам разработчика

3.2.2. Описание логического вида функций системы

Подсистема реализует преценденты использования (usecase) для заинтересованных лиц в соответствии с рис. 2. На каждый usecase с данной диаграммы будет отведен отдельный метод слоя сервиса.

Рис. 2 — Usecase диаграмма подсистемы «Courses»



3.2.3. Продажа доступа к курсам

Проблема и задача: требуется реализовать отказоустойчивый процесс покупки курсов. Пользователь нужно гарантированно выдать курс после оплаты, чтобы не нагружать поддержку (customer support), пользователь также хочет, чтобы это произошло сразу после оплаты.

Контекст: в данной задаче фигурирует и используется подсистема «Finances»

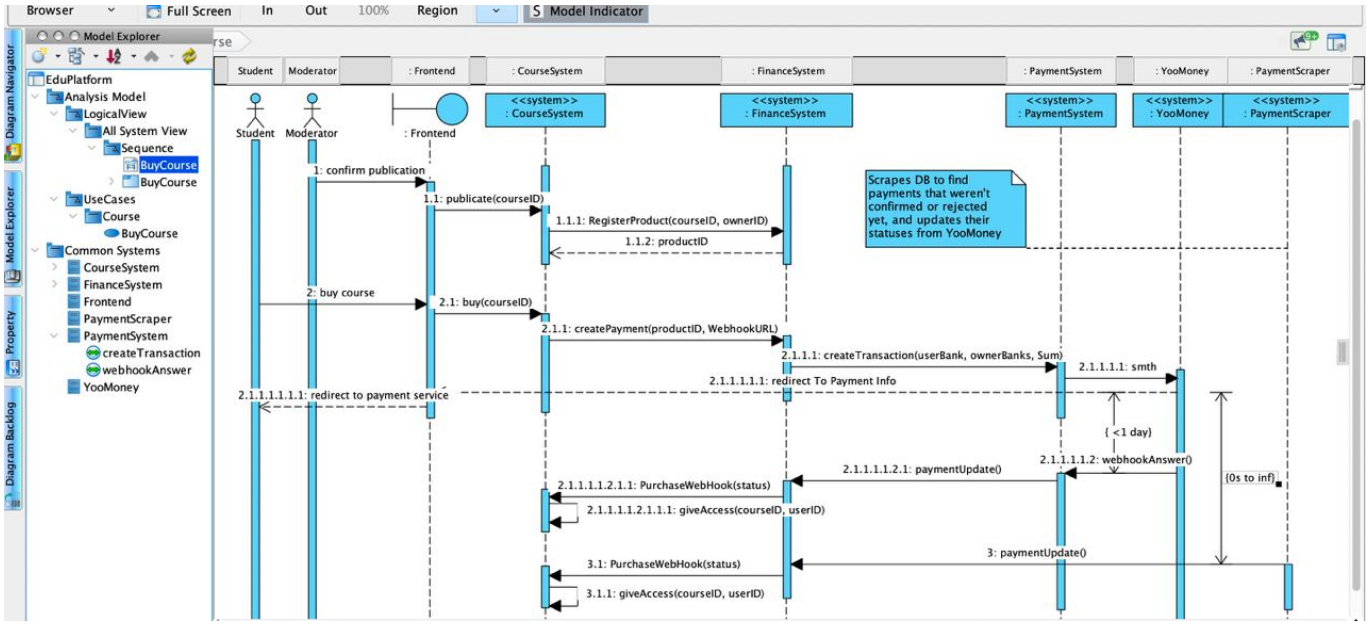
Решение: Решение использует метод взаимодействие webhook (вебхук). А также паттерн Transactional Outbox.

Описание алгоритма: Получив запрос на покупку курса подсистема находит ID продукта этого курса и делает запрос сервису «Finances. Запрос содержит ID аккаунта покупателя, а также url вебхука — адрес, по которому должен сообщить обновление о покупки сервис «Finances». Данный запрос отправляется в сервис «Finances». Подсистема «Finances» формирует запись платежа в базу данных в outer box и ставит ему статус "в обработке данная запись содержит url вебхука. Далее «Finances» формирует информацию о переадресации пользователя на сайт платежной системы и отправляет это ответом подсистеме «Courses», которая переадресует пользователя. После этого демон (daemon, фоновый процесс) периодически читает outer box, находя там платеж со статусом "в обработке"отправляет запрос в платежный сервис и обновляет текущий статус платежа. Если платежный сервис ответил демону что оплата произошла

успешно, то демон меняет статус записи в outer box на "подтверждено". Далее демон по url для вебхука из записи обращается к сервису «Courses», и подтвержает оплату согласно методу взаимодействия webhook. Только если сервис «Courses» успешно ответит демону, он поменяет статус записи в outer box на "обработано в ином случае он снова пройдет на эту запись и снова попытается добиться обработки подтверждения покупки у сервиса «Courses». При этом в сервисе «Finances», также будет API эндпоинт, который будет ждать ответа по вебхуку от платежного сервиса, если он получает его, то отправляет подтверждение оплаты по вебхуку в сервис «Courses», если сервис «Courses» успешно ответил, то сервис «Finances» запишет статус "обработано"в записи платежа в outer box. Важно, что обращение по url вебхука к сервису «Courses» должно быть идемпотентным.

Схему данного алгоритма можно увидеть на рисунке 3.

Рис. 3 — Диаграмма последовательности покупки курса



Факторы:

1) Отказоустойчивость. В ней заинтересован заказчик и пользователь.

2) Нагрузка решения на систему.

3) Сложность решения.

4) Скорость получение доступа к курсу с момента совершения оплаты.

Альтернативы: вместо метода взаимодействия webhook можно использовать retry, который запускает процесс, периодически запрашивающий обновление у сервиса.

Мотивировка: Данный алгоритм разработан для реализации фактора отказоустойчивости и работает даже при сбоях питания на любом его этапе. Таким образом мы гарантируем выдачу доступа к курсу, а также видимость платежных статусов в outer box. Это удовлетворяет фактору 1. За счет того, что сервис «Finances» также обрабатывает событие совершения оплаты и сразу обращается в сервис «Courses» мы делаем выдачу доступ, по сути, сразу после оплаты, что удовлетворяет фактору 4. Однако данное решение сложное и не удовлетворяет фактору 3.

Метод взаимодействия retry не выбран, так как множество покупок с ним увеличат нагрузку на систему.

3.2.4. Передача объектов мониторинга между слоями

Проблема и задача: на трех слоях архитектуры требуется писать логи и трассировку. Где передавать информацию для логирования и трассировки?

Контекст: для логирования нужен настроенный для разработчика и заказчика объект класса logger. Данный объект может (в будущем или в других проектах) отправлять логи на другой сервер, форматировать логи, удаляя из них приватные данные и другие функции. Аналогично для создания трассировок

Решение: Данные объекты будут передаваться через контекст (Context).

Факторы:

1) Уменьшение количество зависимостей.

2) Тестируемость.

3) Легкость реализации.

4) Легкость внесений изменений в части системы.

Альтернативы: Передача в параметрах методов. Хранение объектов логгера глобально.

Мотивировка: Передача таких объектов через контекст позволяет передавать разные объекты реализации мониторинга в разные части системы (допустим в новой части системы мы должны будем отправлять логи на другой сервер, или же вовсе не отправлять, а записывать в определённый файл). Это удовлетворяет фактору 4. Во время тестирования мы также можем легко передать нужный mock (вид заглушки для тестирования) для этих объектов, это удовлетворяет фактору 2. Мы также убираем все зависимости нашей (бизнес) логики приложения от средств мониторинга (фактор 1). Реализация такого метода средней сложности (нейтрально по отношению к фактору 3).

Передача в параметрах методов не выбрано, так как методы касаются бизнес-логики нашего приложения, на них не должны влиять средства мониторинга, в том числе не должны передаваться в параметрах (согласно Чистому коду [22]).

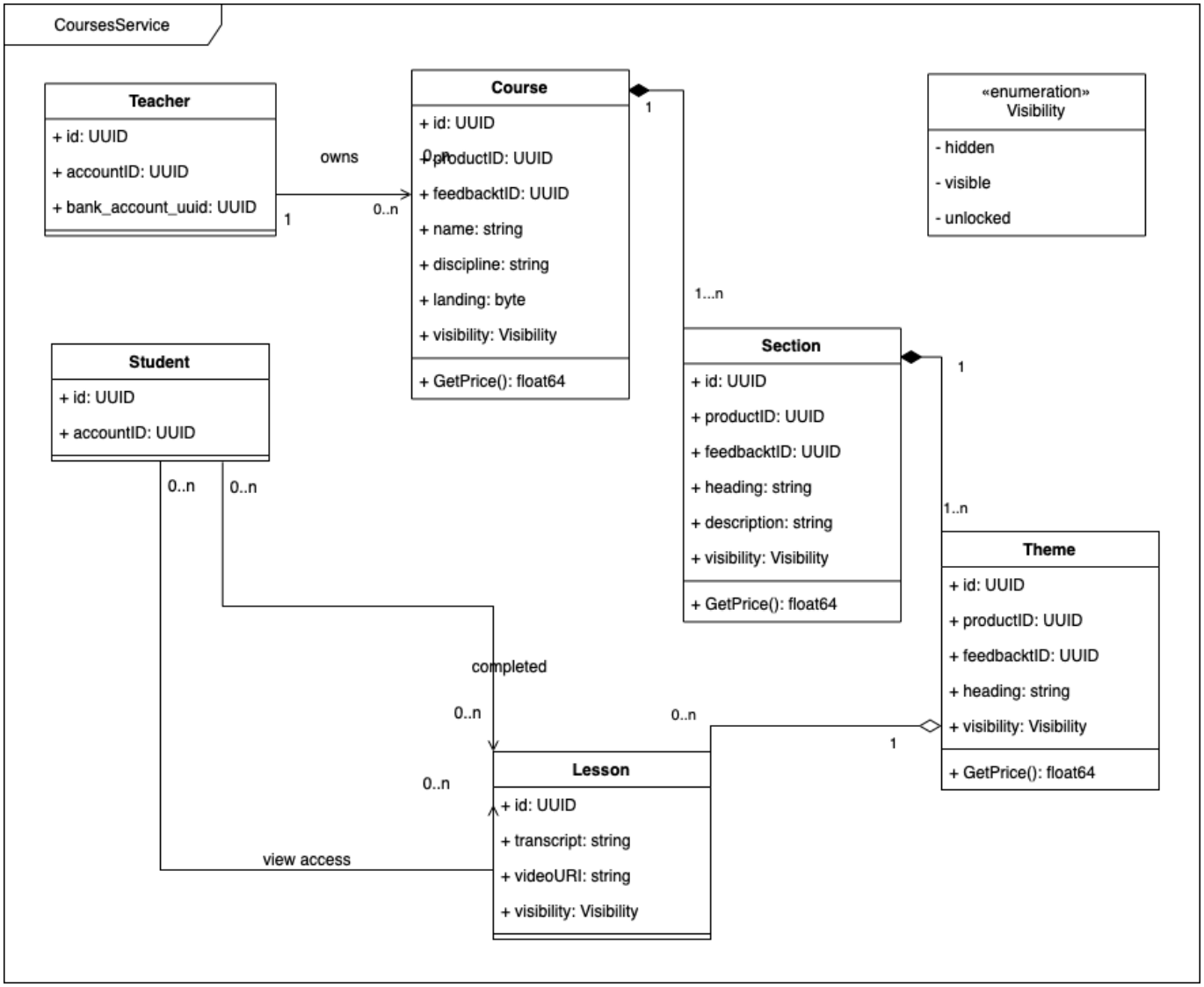
Хранение объектов логгера глобально плохо тестируется (фактор 2) и может иметь создавать проблемы при многопоточном использовании.

3.3. Описание и обоснование архитектуры программы

3.3.1. Описание общего архитектурного решения

Взаимодействие между подсистемами будет вестись асинхронно по протоколу HTTP. Системы серверной части организованы согласно принципам микросервисной архитектуры и разбиты по доменам (DDD, Domain-Driven-Design[24]). Данная программа представляет домен курсов (см рис. 4).

Рис. 4 — Модель предметной области курсов



Программа должна следовать принципам Чистой архитектуры[23]. Данные программы должны быть поделены (зачастую) на три слоя (транспорт, бизнес-логика, репозиторий). Слой транспорт и репозиторий не должны содержать бизнес логики, только получение, агрегация, парсинг и проверка данных.

3.3.2. Архитектура программы

Проблема и задача: требуется выбрать подход проектирования программы позволяющий быстро расширять и изменять ее функционал, реализуя интересы разработчика и заказчика.

Контекст: задаче решается с учетом выбора языка программирования Go

Решение: Программа разрабатывается по принципам Чистой архитектуры.

Факторы:

1) Читаемость и понятность кода.

2) Уменьшение количество зависимостей.

3) Легкость реализации изменений требований.

4) Стандартизация с другими подсистемами.

5) Тестируемость.

6) Легкость внесений изменений в систему.

Альтернативы: Гексагональная архитектура, Domain-Driven-Design[24] (DDD).

Мотивировка: Чистая архитектура распространённый и принятый в Go подход и концепция, что обеспечит разработчикам на Go легкость внесения изменений, а также читаемость и понятность кода. Данный подход известен и непротиворечив, а также описан в книге[23], рекомендованной к прочтению в сфере разработки – это будет способствовать поддержанию стандартизации программы по отношению к программам других будущих и существующих микросервисов. Данный подход включает в себя принципы изоляции зависимостей, позволяющие отделить реализацию бизнес-логики программы от фреймворков, это позволит легко менять основной функционал программы, чтобы прейти на новые требования заказчика. Также данный подход

Чистая архитектура — это надстройка над гексагональной, удовлетворяющая стандартизации больше, чем гексагональная, за счет книги Чистая архитектура. DDD не выбрано, так как это менее распространенный и определённый подход что помешает фактору стандартизации, более того, разделение на основные домены, согласно этому принципу, осуществилось при делении системы на микросервисы.

3.4. Описание и обоснование выбора метода организации входных и выходных данных

3.4.1. Транспортный протокол

Для организации входных и выходных данных в подсистеме используется стандарт HTTP и спецификация к нему.

Факторы:

1) Легкость реализации.
2) Скорость передачи данных по сети.
3) Легкость интеграции (использования) подсистемы.
4) Стандартизация с другими подсистемами.

Альтернативы: GRPC.

Мотивировка: Выбор HTTP обусловлен его широкой поддержкой, легкостью реализации и интеграции с различными клиентскими приложениями (веб , мобильные, десктопные). Поскольку данную подсистему использую только системы конечных пользователей, то скорость и объем передачи данных будет также удовлетворять постановке задачи.

GRPC не выбран из-за того, что его сложнее интегрировать с различными клиентскими приложениями (фактор 3).

3.5. Описание и обоснование выбора состава технических и программных средств

3.5.1. Язык программирования

Проблема и задача: выбор языка программирования для реализации подсистемы «Courses»

Контекст: согласно постановке задачи подсистема «Courses» является микросервисом (в системе Образовательной платформы, спроектированной по принципам микросервисной архитектуры)

Решение: Выбран язык программирования Go.

Факторы:

1) Успешная обработка одновременных запросов к подсистеме при резком повышении их количества. (это может произойти, если преподаватель начал урок, и большой поток участников курса находит на платформу одновременно).
2) Кол-во требуемых процессоров для обработки одновременных запросов. Это влияет на стоимость аренды серверов.
3) Стандартизация с другими подсистемами.
4) Читаемость и понятность кода.
5) Скорость разработки.

Альтернативы: Java, Python, Erlang

Мотивировка: Go был разработан в Google в 2007, его разработка ориентировалась на использование языка для создания высокопроизводительных и масштабируемых серверных приложений, веб-сервисов и других распределённых приложений.

Опишем уникальные технические преимущества данного языка – его модель многопоточности. Данные преимущества отражаются в основном эффективностью и производительностью в обработке конкурентных запросов. Данные преимущества Go созданы уникальной моделью многопоточности и встроенным средствам многопоточности (goroutines или горутины). Горутина это легковесный поток, который позволяет выполнение конкурентных задач в программе на Go. Данными горутинами в приложении управляет Go runtime – это часть исполнительной среды Go, которая отвечает за выполнение программы (управление памятью, планирование горутин, сборку мусора, системные вызовы). Управление горутинами в Go runtime, включает распределение ресурсов процессоров операционной системы по горутинам, и реализовано согласно M:N модели многопоточности. Данная модель в Go реализует эффективное распределение выполнения M горутин по N потокам операционной системы (далее – системные потоки). Кратко опишем как это происходит данное распределение. Go runtime создает логические процессоры в количестве P (можно настроить в конфигурации программы), зачастую P соответствует количеству системных потоков, которые мы хотим использовать. Каждому логическому процессору выделяется своя локальная очередь горутин и системный поток. Все горутины в программе в количестве M помещаются в глобальную очередь. Логический процессор раз в 61 итерацию забирает себе в локальную очередь M/P (свою долю) первых горутин из глобальной очереди. В остальные итерации он за промежуток процессорного времени (далее time slice) выполняет первую горутину из своей локальной очереди, и, если она не успела завершиться – ставит ее в конец глобальной очереди. Если у логического процессора нету горутин в локальной очереди, и нету в глобальной – он забирает половину горутин у другого процессора. Если горутина делает системный вызов и блокирует системный поток, от логического процессова откреплляется данный поток и присваивается новый. Иллюстрации и описание на наглиском языке можно посмотреть в статье[25].

За счет данной модели распределения конкурентной нагрузки, реализованной в самой программе, а встроенными средствами операционной системы, а также оптимизированных быстрых вызовов средств операционной системы и системных потоков (за счет совместимости с языком C), Go с его runtime может эффективно обрабатывать одновременные запросы в количествах в разы больших чем другие языки. Данное преимущество важно для факторов 1 и 2. Другие языки имеют модели многопоточности 1:1 (Python) и 1:M (Java, распределяет M потоков программы на один системный), которые не позволят равномерно распределять задачи собственных потоков по всем доступным системным.

Единственным языком, в котором имеется схожая с Go модель многопоточности является язык Erlang, однако данный язык не подойдет согласно фактору 4, так как он является функциональным, и более сложным в освоении, чтении и найме новых разработчиков.

В идиоматику (общие принципы, основные на культуре разработки, создании языка и его библиотек) Go входят принципы, соответствующие факторам 3 и 4. Такие как явность передачи переменных, возвращения ошибок в коде, а также использования одинаковых больших open source библиотек.

В рамках данного проекта начало разработки на Go будет медленным в связи с тем, что в Go нет используемых фреймворков для быстрого создания веб приложений или же они противоречат идиоматике (например использования ORM для работы с базой данных). Однако долгое создания основы (шаблона) приложений на Go, компенсируется повышенной скоростью сопровождения всех написанных сервисов, за счет того, что разработчики знают реализацию и

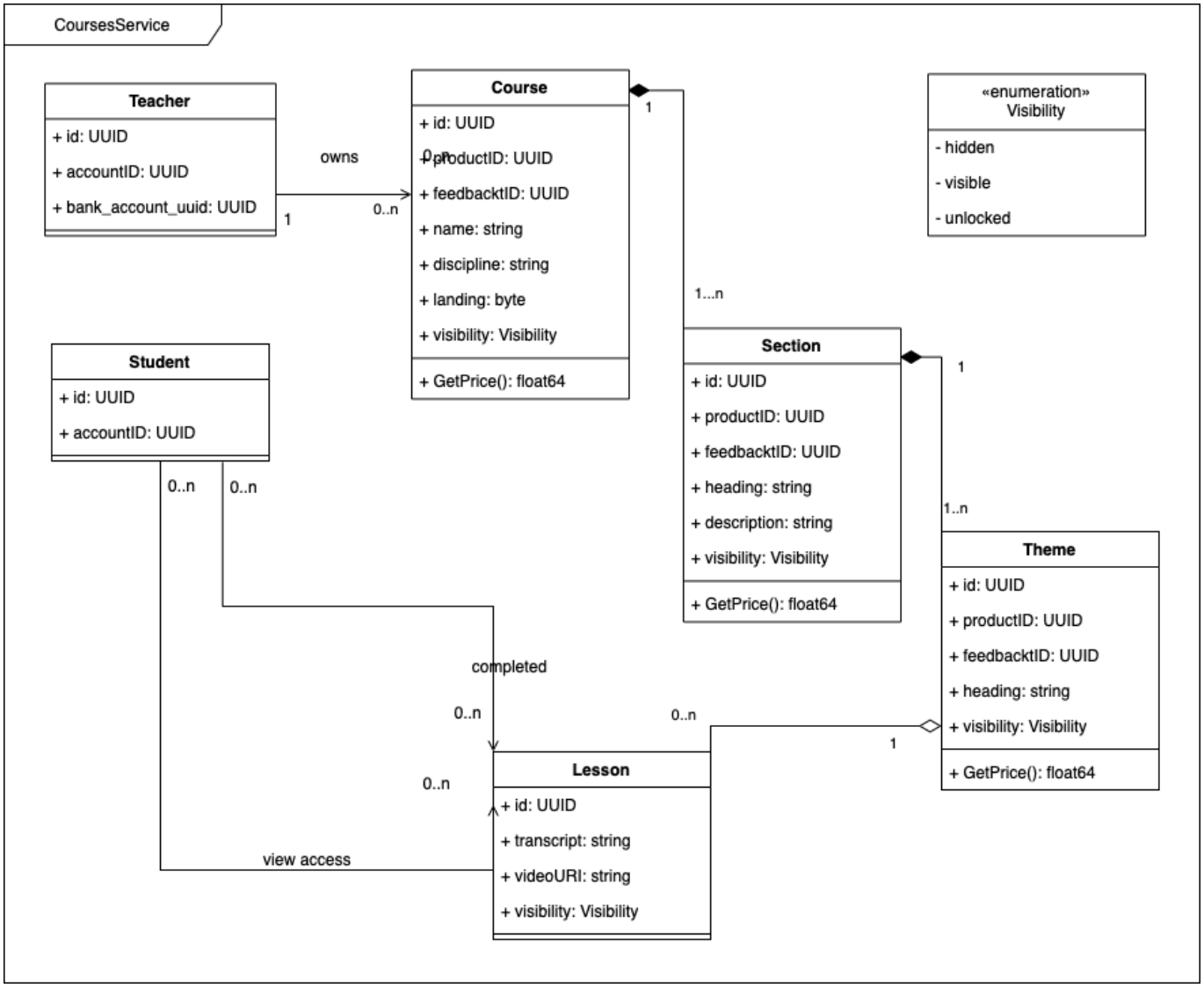
детали работы всех компонент программы. Данная особенность нейтрально влияет на фактор 5.

3.5.2. База данных и СУБД

Проблема и задача: выбор базы данных для хранения информации о курсах

Контекст: Чтение курса или определённого урока будет происходить часто согласно постановке задачи. Курсы и их элементы будут не часто меняться. То есть малая нагрузка на запись (согласно постановке задачи). Состав курса в соответствии с моделью предметной области (см. рис. 5).

Рис. 5 — Схема данных курса



Решение: Выбрана MongoDB. Скелет, структура курса (курс содержащий секции и темы, но без сущности Lesson) будет хранится в коллекции «Courses». В данной стуктуре в элементах Theme будут содержаться массивы содержащие id и название объекта Lesson принадлежащего соотвествующей Theme. Объекты Lesson будут храниться в отдельной коллекции «Lessons».

Факторы:

1) Поддержка гибкой схемы. Схема данных объекта может поменяться согласно изменению требований, который происходят часто в соответствии с интересом заказчика.

2) Чтение курса или определённого урока должно быть быстрым.

3) Скорость разработки.

Альтернативы: PostgresQL, хранить сущности в 4 соответствующих таблицах

Мотивировка: Используя такое решение мы сможем быстро выдавать все элементы для навигации пользователя по частям курса в виде объекта из коллекции «Courses». При этом, когда пользователю нужен будет посмотреть интересный ему урок, он получит его одним запросом из коллекции «Lessons».

MongoDB использует документо-ориентированную модель данных, хранит данные в документах подобные JSON-объектам, но в бинарной кодировке (bson). За счет этого можно эффективно и легко работать с сущностями, имеющими внутри себя большую вложенность. Это удовлетворяет фактору 3. Также это будет хорошим вариантом для поддержания гибкой схемы — после изменения схемы старые записи, курсы, можно оставить какими они есть, а новые курсы добавлять и проверять на соответствие новой схеме. Также в MongoDB можно удобно работать с опциональными полями. Это удовлетворяет фактору 1.

В PostgresQL более затруднительно поддерживать гибкую схему и опциональные поля, что не будет упрощать разработку. В PostgresQL также на выдачу скелета курса, нужно было бы делать операцию JOIN на 4 таблице, что медленнее чем взять один элемент в MongoDB. Поэтому PostgresQL не был выбран.

4. Ожидаемые технико-экономические показатели

4.1. Общие требования к технико-экономическим показателям

Требований об технико-экономических показателей и характеристик и их учета не ставятся.

Рекомендовано разработать программный продукт, адаптированный для имплементации бизнес метрик, служащих для будущего проведения анализа эффективности продукта и валидации иформационной системы.

4.2. Сравнительная таблица

Для большего понимания функционального состава продукта и его различия с аналогами приведена таблица сравнения с наиболее популярными в России и за рубежом сайтами: Magisteria[17], Stepik[18], Coursera[19], Skillbox[20], Яндекс Образование[21].

1 - Образовательная платформа
2 - Magisteria
3 - Stepik
4 - Coursera
5 - Яндекс Образование
6 - Skillbox

Функционал	1	2	3	4	5	6
Создание курсов	+	-	+	+	-	-
Приобретение курсов	+	+	+	+	+	+
Отзывы	+	+	-	+	+	+
Генерация чатов	+	-	-	-	-	-
Доступен в России	+	+	+	-	+	+

5. Список используемой литературы

1) ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

2) ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

3) ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

4) ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

5) ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

7) ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

8) ГОСТ 19.603-78 Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

9) ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

10)ГОСТ 19.404-79 Программа и методика испытаний. Требования к содержанию и оформлений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

11) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

12) ГОСТ 19.505-79 Руководство оператора. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

13) ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

14)ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

15) c4model, [Электронный ресурс] Режим доступа: `https://c4model.com`.

17) Magisteria, [Электронный ресурс] Режим доступа: `https://magisteria.ru`.

18) Stepik, [Электронный ресурс] Режим доступа: `https://stepik.org/`.

19) Coursera, [Электронный ресурс] Режим доступа: `https://www.coursera.org/`.

20) Skillbox, [Электронный ресурс] Режим доступа: `https://skillbox.ru`.

21) Яндекс Образование, [Электронный ресурс] Режим доступа: `https://education.yandex.ru`.

22) Martin, R. C. Clean Code: A Handbook of Agile Software Craftsmanship. — Prentice Hall, 2008. — 464 с.

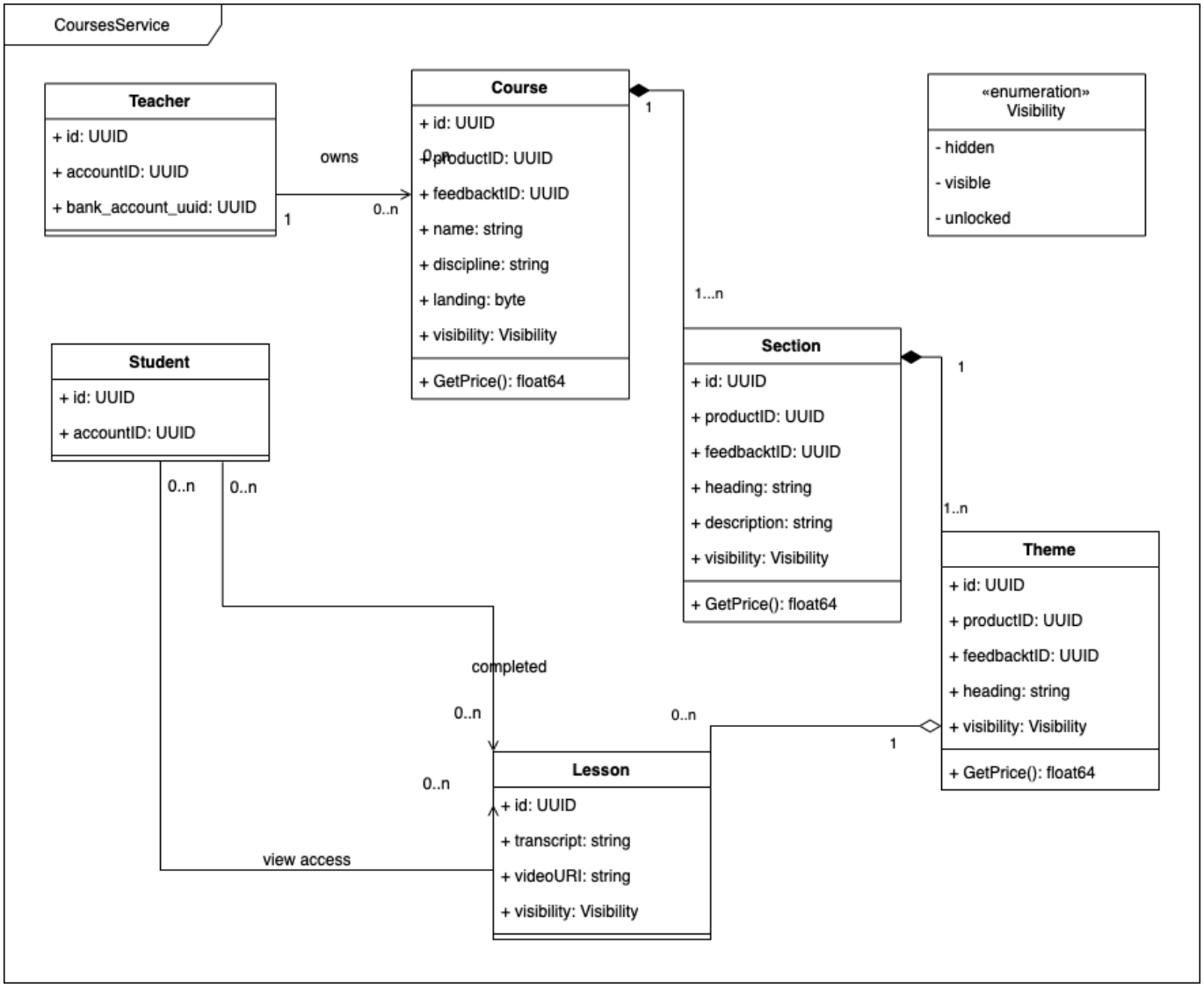
23) Martin, R. C. Чистая архитектура. — Питер, 2018. — 432 с.

24) Эванс, Э. Domain-Driven Design: Tackling Complexity in the Heart of Software. — Addison-Wesley, 2003. — 560 с.

25) Kelche, K. The Golang Scheduler [Электронный ресурс]. — `https://www.kelche.co/blog/go/golang-scheduling`. — Дата доступа: 25 февраля 2024 г.

Приложение 1. Модель предметной области подсистемы Courses

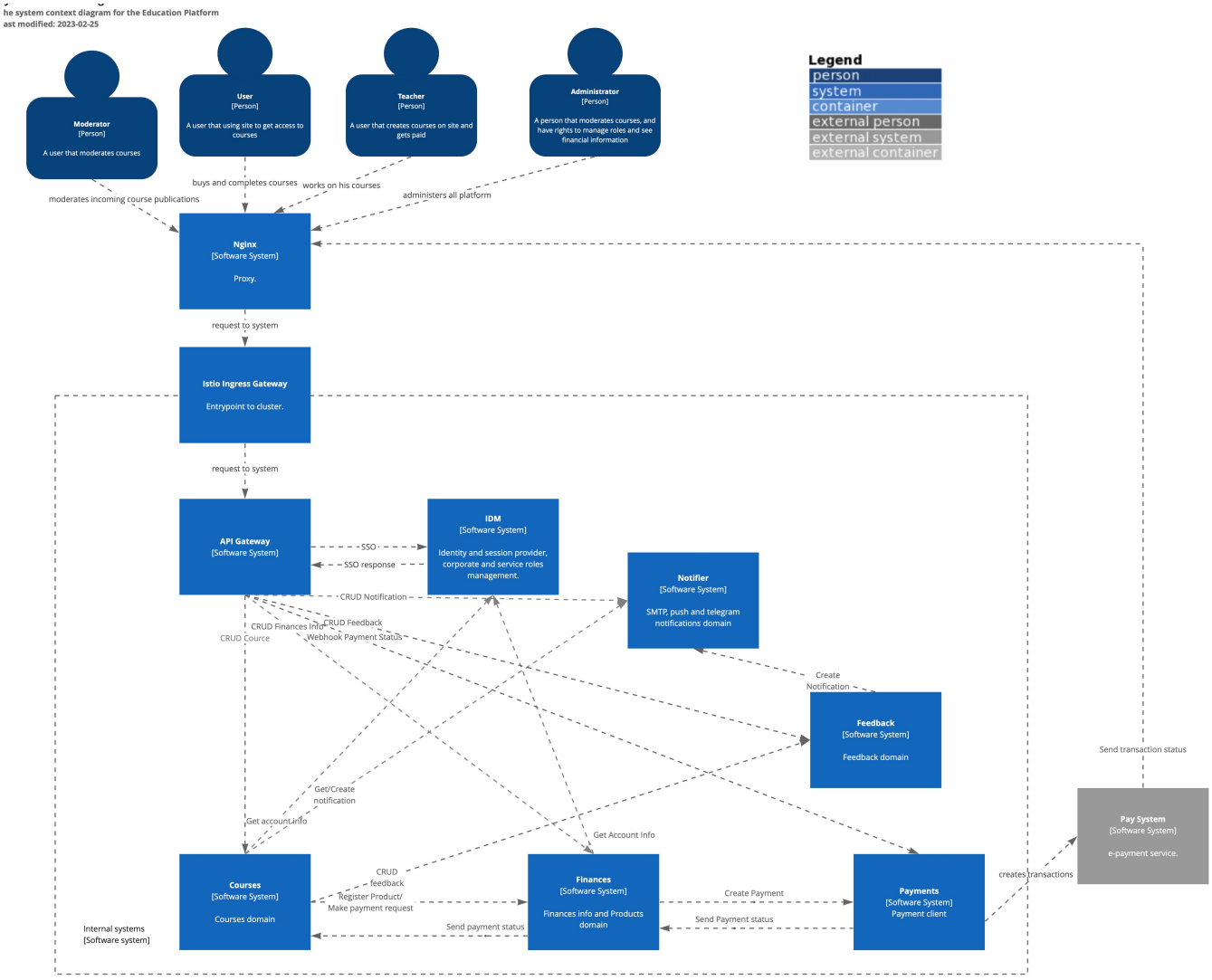
Рис. 6 — UML diagram. Courses system



Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.09.12-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Приложение 2. Уровень контекста системы Edu Platform

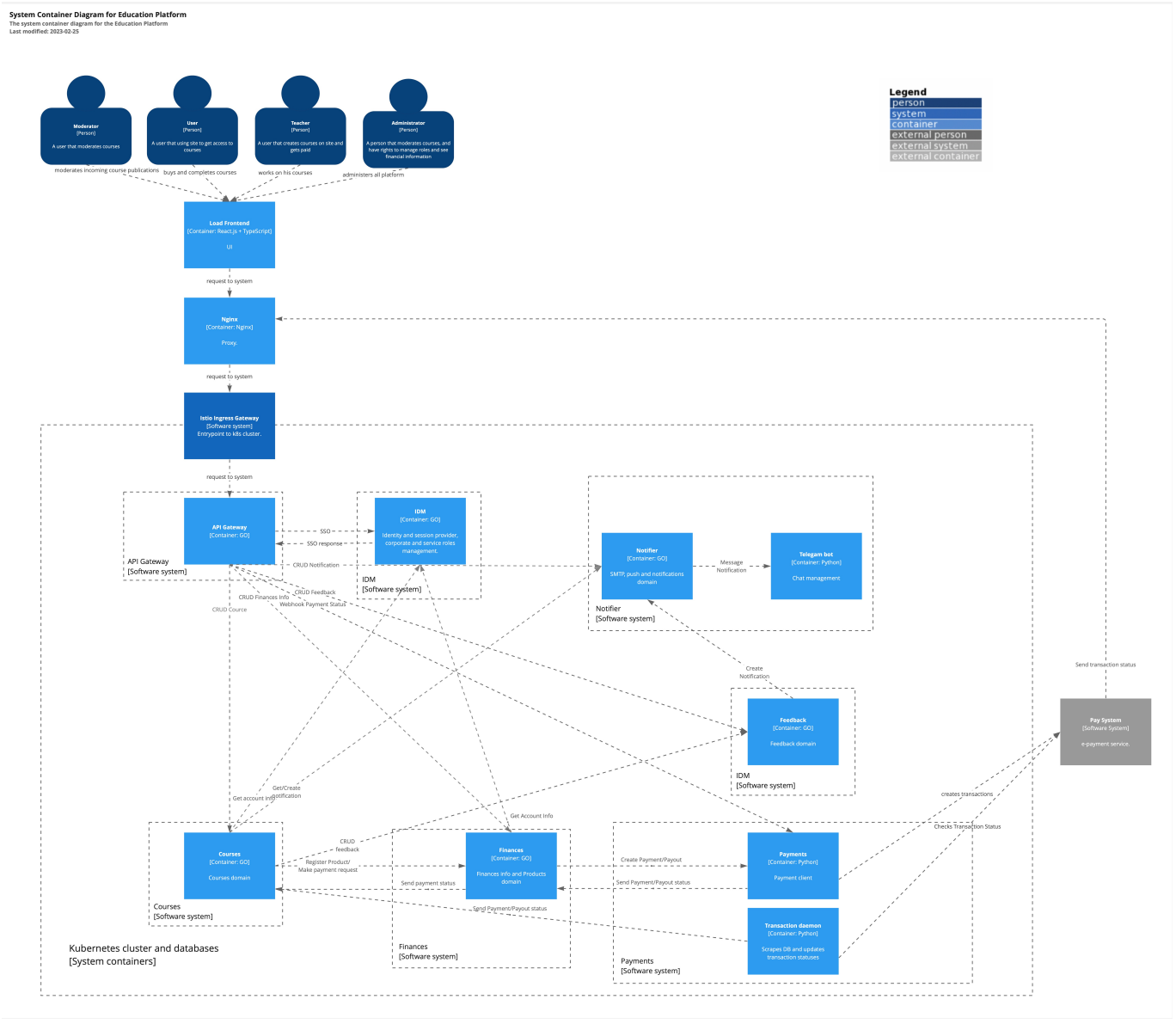
Рис. 7 — Context level diagram. Edu Platform c4 model



Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.09.12-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

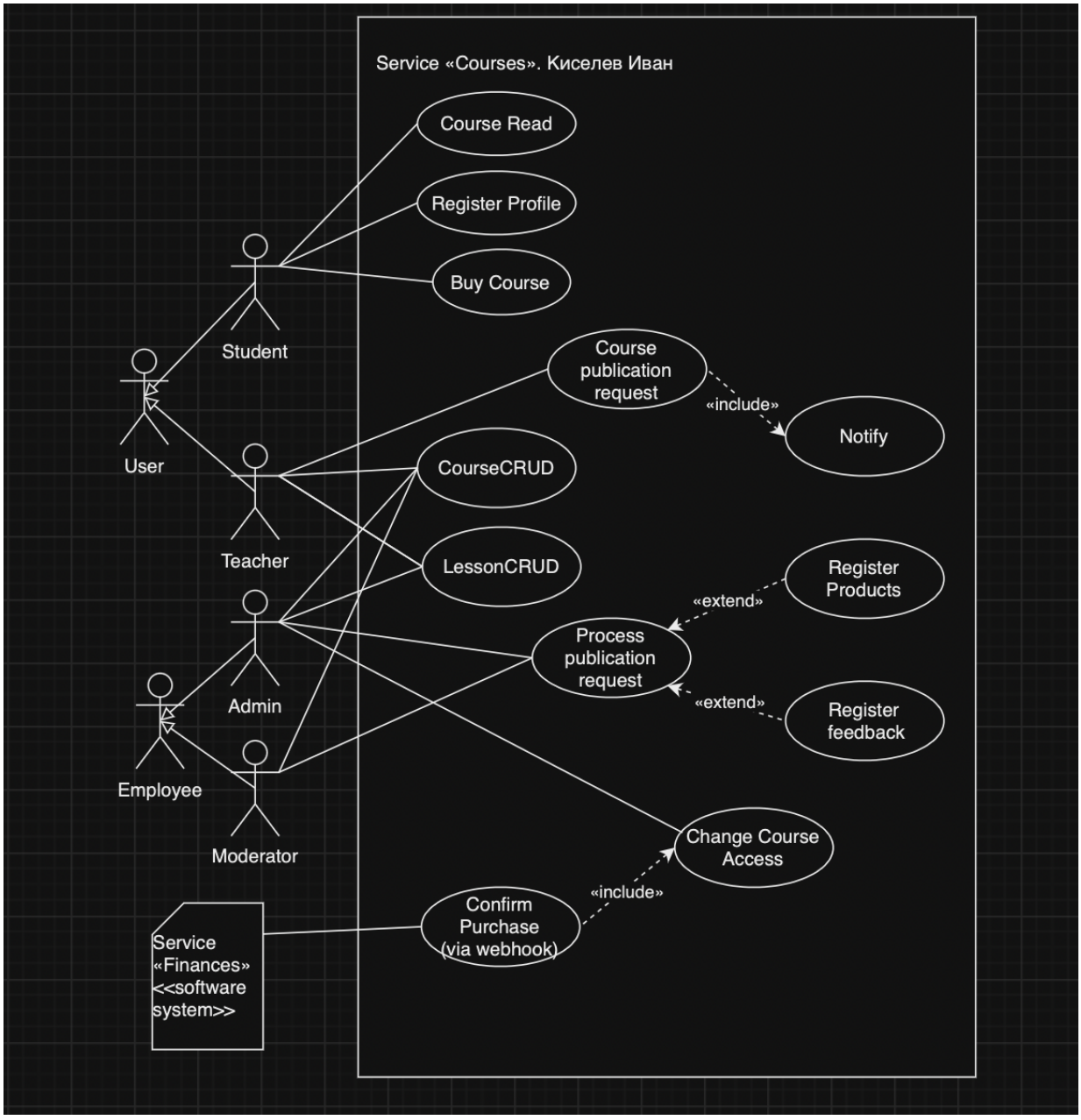
Приложение 3. Уровень контейнера системы Edu Platform

Рис. 8 — Container level diagram. Edu Platform c4 model



Приложение 4. Модель прецендентов подсистемы Courses

Рис. 9 — Usecase diagram. Courses software system



6. Глоссарий

1) Конкурентность - разделение каким-то процессом общих ограниченных ресурсов с другими процессами. Зачастую таким ресурсом является единица процессорного времени.

2) Multistage Build (Многоэтапная сборка) - методология сборки программного обеспечения, которая включает несколько этапов, где каждый использует определенные инструменты и зависимости для выполнения его задач. Этот подход позволяет уменьшить размер конечного образа или бинарного файла, улучшить скорость сборки и обеспечить более чистое и эффективное управление зависимостями и окружением исполняемого файла.

3) API (Application Programming Interface) – Интерфейс программирования приложений, набор соглашений и инструментов, которые позволяют различным программным компонентам взаимодействовать друг с другом. API определяет методы и структуры данных, которые могут быть использованы для выполнения определенных задач или обмена информацией между программами.

4) Docker – Платформа для контейнеризации приложений, которая позволяет упаковывать приложения и их зависимости в легко переносимые контейнеры. Docker обеспечивает изоляцию ресурсов и среды выполнения.

5) Docker Compose – Инструмент для определения и управления многоконтейнерными приложениями с помощью файла конфигурации в формате ".yaml". Docker Compose позволяет определить сервисы, их зависимости и настройки в одном файле, что упрощает развертывание и управление многоконтейнерными приложениями.

6) URL (Uniform Resource Locator) – Единый указатель ресурса - адрес, который определяет местоположение ресурса в сети Интернет. URL состоит из протокола, доменного имени и пути к ресурсу.

7) СУБД (Система управления базами данных) - программное обеспечения для использования баз данных.

8) Микросервис - программная подсистема и единица Микросервисной архитектуры программного обеспечения, направленной на разбиение системы на атомарные слабо связанные и легко изменяемые модули.

9) MongoDB — документоориентированная система управления базами данных.

10) Postman — это приложение для тестирования API.

11) C4 Model - методология архитектуры ПО, которая позволяет описывать систему на различных уровнях абстракции (контекст, контейнеры, компоненты и код)

Лист регистрации изменений

Изм.	Номера листов				Всего листов в документе	№ документа	Входящий № сопровод. докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					