

Edu platform

Архитектурный документ

Группа: БПИ217

Авторы документа:

Киселев Иван (Разделы и диаграммы системы Courses, стандартизация платформы, противодействие атакам, требования к безопасности, c4 model Context, обоснование выбора Go и методов связанных с ним)

Команда:

Фёдоров Артём

Мохов Сергей

0. Раздел регистрации изменений



Версия документа	Дата изменения	Описание изменения	Автор изменения
1.0.0			
1.0.1			

1. Введение

1.1. Название проекта

Образовательная платформа.

1.2. Наименование проекта на английском языке

Educational platform.

1.3. Задействованные архитектурные представления

1. Представление прецедентов (Use Case Diagram, User Story Mapping, табличное описание Business case, сценарий использования)
2. Логическое представление (Диаграмма предметной области, C4 Context model, Sequence Diagram, диаграмма классов, обозначение пакетов и их организации и разделения)
3. Представление сетей
4. Представление интеграции kubernetes

(вместо страниц описал в пунктах)

- C4 model дает гибкие возможности проектирования подходящие методологии agile.
- Аналогично User Story Mapping, позволял работать с заказчиком. Этот метод помогает разбить функции на подсистемы и команды.
- Sequence диаграмма нужна чтобы показать работу с в рамках выбранной чистой архитектуры
- Диаграмма классов показывает обязательное требование к содержанию моделей в домене микросервиса Courses
- Представление сетей – сетевая безопасность
- Представление интеграции kubernetes – обоснование времени доступности сервисов

1.4. 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.

1.5. Рамки и цели проекта

Проект "Образовательная платформа" включает в себя следующие системы:

1. Фронтенд - обеспечивает интерфейс взаимодействия с пользователями, разрабатывается на React.js + TypeScript.
2. Gateway - является промежуточным слоем между фронтеном и бэкендом, обеспечивает маршрутизацию запросов, разрабатывается на Go.
3. IDM (Identity Manager): Отвечает за управление и аутентификацию пользователей, разрабатывается на Go.
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 Go.
8. Feedback: Модуль для сбора обратной связи от пользователей, разрабатывается на Go.
9. Telegram Bot: Позволяет взаимодействовать с платформой через Telegram, разрабатывается на Python.

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 посыпать 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 расширить 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 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. Архитектурные факторы

2.1. Ключевые заинтересованные лица

Действующее лицо	Заинтересованность в системе
Разработчик	Простота поддержки и введения новой функциональности
	Простота изучения и понимания строения проекта
	Простота нахождения ошибок
Учитель	Простота добавления своих курсов в систему и вывод средств с их продажи
Ученик	Простота изучения курсов и процесса покупки
Администратор	Простота отслеживания статусов и нагрузки системы
	Контроль над корпоративными частями системы (роли, ошибки, жалобы)
Незарегистрированный пользователь	Простота и удобство получения информации о курсах и ценах

2.2. Ключевые требования к системе

Функциональное назначение

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.

Клиент, получив доступ, сможет размещать курсы и получить роль "учитель". Данный 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 солями 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 иолями пользователей.

Эксплуатационное назначение (использование информационной системы):

Данная система будет представлена в виде нескольких подсистем. Часть подсистем будет реализовать клиентскую часть, веб-браузер с программой, обрабатывающей действия пользователя на веб-сайте. Клиентская часть будет выводить визуализировать необходимые данные пользователь

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 запущены в контейнерах на нескольких серверах. Данные подсистемы будут принимать запросы по сети от друг друга или от клиентской части. Подсистемы и связь между ними описана в пункте 4.

Нефункциональные требования

1. Производительность: Система должна обеспечивать быструю загрузку страниц и обработку запросов пользователей.
2. Масштабируемость: Возможность масштабирования системы для поддержки роста количества пользователей и курсов.
3. Гибкость и расширяемость: Легкость внесения изменений и добавления новой функциональности.

Серверная часть должна быть устойчива к высоким нагрузкам – целевое значение доступности должно составлять не менее 80% времени.

Рекомендуется разрабатывать системы с расчетом на высокую нагрузку на “чтение”, для системы Payments и части системы Notifier отвечающую за отправку сообщений в телеграм на ближайшие этапы разработки сильных нагрузок (RPS) не ожидается.

Остальные требования описаны в [последней редакции “Техническое задание Образовательная платформа”](#) на момент последнего изменения данного документа.

Со стороны безопасности система не должна быть уязвима к XSS, CSRF, атаки по взлому паролей, RCE, и атаки на сети (DDOS).

2.3. Ключевые ограничения

1. Технические ограничения (к железу и поддержке систем):
 - 1.1. Фронтенд - React
 - 1.2. Gateway - язык Go версии 1.21
 - 1.3. IDM (Identity Manager) - язык Go версии 1.21
 - 1.4. Courses - язык Go версии 1.21
 - 1.5. Finances - язык Go версии 1.21
 - 1.6. Payments - язык Python версии 3.10
 - 1.7. Notifier - язык Go версии 1.21
 - 1.8. Feedback - язык Go версии 1.21
 - 1.9. Telegram Bot - язык Python версии 3.10
 - 1.10. Database IDM - СУБД PostgreSQL 16
 - 1.11. Database Finances - СУБД PostgreSQL 16
 - 1.12. Database Courses - СУБД MongoDB 7.0
 - 1.13. Database Notifier - СУБД MongoDB 7.0
 - 1.14. Database Feedback - СУБД MongoDB 7.0
2. Организационные ограничения:
 - 2.1. Ограниченные ресурсы на разработку и поддержку проекта (в рамках этого документа точные ограничения не учитываются).
 - 2.2. Необходимость соблюдения сроков и бюджета проекта (в рамках этого документа точные ограничения не учитываются).

3. Другие ограничения

- 3.1. Система должна поддерживать сбор метрик
- 3.2. Система должна соответствовать нефункциональным требованиям

3. Общее архитектурное решение

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:

10. Frontend - обеспечивает интерфейс взаимодействия с пользователями, разрабатывается на React.js + TypeScript.
11. Gateway - является промежуточным слоем между фронтендом и бэкендом, обеспечивает маршрутизацию запросов, разрабатывается на Go.
12. IDM (Identity Manager): Отвечает за управление и аутентификацию пользователей, разрабатывается на Go.
13. Courses: Бэкенд для управления курсами, разрабатывается на Go.
14. Finances: Управление финансовыми транзакциями и платежами, разрабатывается на Go.
15. Payments: Отвечает за взаимодействие с платежными системами, разрабатывается на Python.
16. Notifier: Обеспечивает отправку уведомлений пользователям, разрабатывается на Go.
17. Feedback: Модуль для сбора обратной связи от пользователей, разрабатывается на Go.
18. Telegram Bot (часть системы Feedback): Позволяет взаимодействовать с платформой через Telegram, разрабатывается на Python.
19. Системы для реализации мониторинга, трассировок (Prometheus, Grafana, Open Telemetry + Jaeger)
20. 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 автоматизированной системы).

3.1. Принципы проектирования

Разработчики программ на языке GO должны следовать принципам чистой архитектуры (или как минимум гексагональной архитектуры с отдельным согласованием). Данные программы должны быть поделены (зачастую) на три слоя (транспорт, бизнес-логика, репозиторий)

Важные аспекты на которые мы ориентируемся:

- Слой транспорт и репозиторий не должны содержать бизнес логики, только получение, агрегация, парсинг и проверка данных.
- Принцип инверсии зависимостей
- Все данные нужные для реализации бизнес кейса (особенно между слоями) должны передаваться в параметрах и результатах функций (сквозь слои не должны тянуться данные неявно, “цыганской ниткой”), допускается лишь неявная передача информации для мониторинга, трассировки и логирования.

Также от кода ожидается качественная декомпозиция, cohesion, хороший нейминг и самое важное – соблюдение принципов SOLID. Для корректной организации кода согласно таким принципам рекомендуется к прочтению “Чистый Код” Роберта Мартина.

Об организации стандартизации на платформе. Для микросервисов на Go создан общий стандартизованный шаблон, начиная с которого будут разрабатываться сервисы, данный шаблон для стандартизации будет рекомендацией к структуре пакетов в приложении, а также будет стандартом для запуска приложения, мониторинга и логирования. Каждый сервис также будет содержать манифест с важной информацией о нем.

О самой платформе. Платформа рассматривается на дальнейший интерес по интеграции корпоративных приложений, поэтому каждый сервис должен быть рассчитан на то, что его будут использовать и другие будущие платформы. Отсюда и происходит решение на разбиение на независимые сервисы, те которые не знают друг о друге.

В данном документе не рассматривается решения для проектирования sag и workflow для процессов связанных с несколькими микросервисами. Однако будет ожидаться их реализация через событийную схему и, вероятно, очередь сообщений (например Kafka).

Принципы SOLID и другие:

1. Принцип разделения ответственности (*Separation of Concerns*):

Каждый компонент системы должен быть ответственен только за определенный аспект функциональности. Например, фронтенд отвечает за пользовательский интерфейс, а бэкенд за бизнес-логику и доступ к данным. Это обеспечивает легкость в сопровождении и модификации системы.

2. Принцип единственной ответственности (*Single Responsibility Principle*):

Каждый модуль или класс должен иметь только одну причину для изменений. Например, компоненты управления пользователями и управления курсами в бэкенде должны быть разделены для избежания излишней сложности и повышения гибкости системы.

3. Принцип инверсии зависимостей (*Dependency Inversion Principle*):

Модули верхнего уровня не должны зависеть от модулей нижнего уровня, а должны зависеть от абстракций. Например, компонент уведомлений не должен зависеть напрямую от конкретной реализации отправки сообщений, а должен зависеть от интерфейса уведомлений, что позволяет легко изменять способы отправки уведомлений без изменения самого компонента.

4. Принцип открытости/закрытости (*Open/Closed Principle*):

Система должна быть открыта для расширения, но закрыта для модификации. Например, новые типы курсов могут быть добавлены без изменения существующего кода.

5. Принцип минимальности (*Keep It Simple, Stupid*):

Система должна быть максимально простой и понятной. Избегайте излишней сложности и лишних элементов, которые могут усложнить сопровождение и разработку.

6. Принцип независимости от фреймворков (*Independence of Frameworks*):

Бизнес-логика системы должна быть независима от каких-либо фреймворков. Это обеспечивает возможность легко заменять или обновлять фреймворки без изменения бизнес-логики.

4. Архитектурные представления

4.1. Представление прецедентов

Здесь приведены бизнес цели платформы и исходящие из них бизнес и use кейсы.

Цель (автоматизированной) системы

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.

Клиент, получив доступ, сможет размещать курсы и получит роль "учитель". 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 посыпать рассылки на почты учителей и участникам курсов.

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

Использование информационной системы:

Данная система будет представлена в виде нескольких подсистем. Часть подсистем будет реализовать клиентскую часть, веб-браузер с программой, обрабатывающей действия пользователя на веб-сайте. Клиентская часть будет выводить визуализировать необходимые данные пользователь 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 запущены в контейнерах на одном или нескольких серверах. Данные подсистемы будут принимать запросы по сети от друг друга или от клиентской части. Подсистемы и связь между ними описана в пункте 4.

Актёры:

Пользователь сервиса курсов:

- Учитель
- Ученик
- Модератор

Пользователь платформы:

- Пользователь (включая неавторизованного)
- Администратор

Бизнес кейсы:

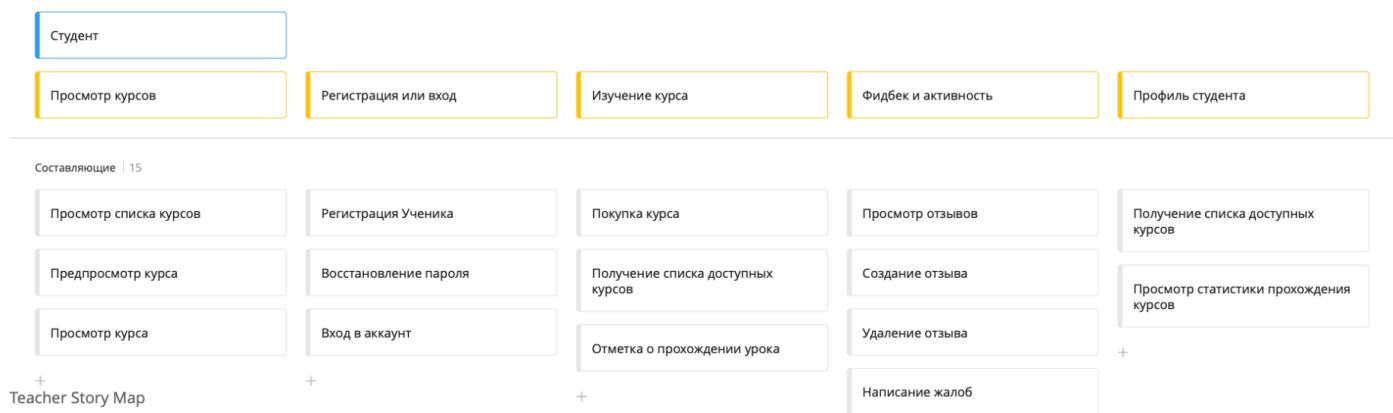
Были выделены (методом User Story Mapping). Данные бизнес кейсы для удобства описаны (и смоделированы) в виде User Story Mapping, а также таблиц разбиения на системы ниже

- 1) Регистрация Учителя
- 2) Регистрация Ученика
- 3) Регистрация Модератора
- 4) Вход в аккаунт
- 5) Восстановление пароля
- 6) Блокировка аккаунта
- 7) Изменение данных аккаунта
- 8) Создание группы ролей
- 9) Создание роли для группы
- 10) Просмотр аккаунта на платформе
- 11) Предпросмотр курса
- 12) Просмотр курса
- 13) Просмотр списка курсов
- 14) Получение списка доступных курсов
- 15) Покупка курса
- 16) Выплата накоплений
- 17) Создание шаблона курса
- 18) Изменение шаблона курса
- 19) Заявка на публикацию курса
- 20) Обработка заявки на публикацию курса
- 21) Публикация курса
- 22) Отметка о прохождении урока
- 23) Просмотр статистики курса
- 24) Просмотр статистики прохождения курсов
- 25) Просмотр истории финансовых транзакций
- 26) Создать скидку
- 27) Завершить скидку
- 28) Просмотр отзывов
- 29) Создание отзыва
- 30) Удаление отзыва
- 31) Отправка уведомления на платформу
- 32) Прочтение уведомления на платформе
- 33) Отправка электронного письма на почтовый адрес
- 34) Отправка сообщения в Telegram чат
- 35) Обновление информации в закрепленном сообщении в Telegram чате

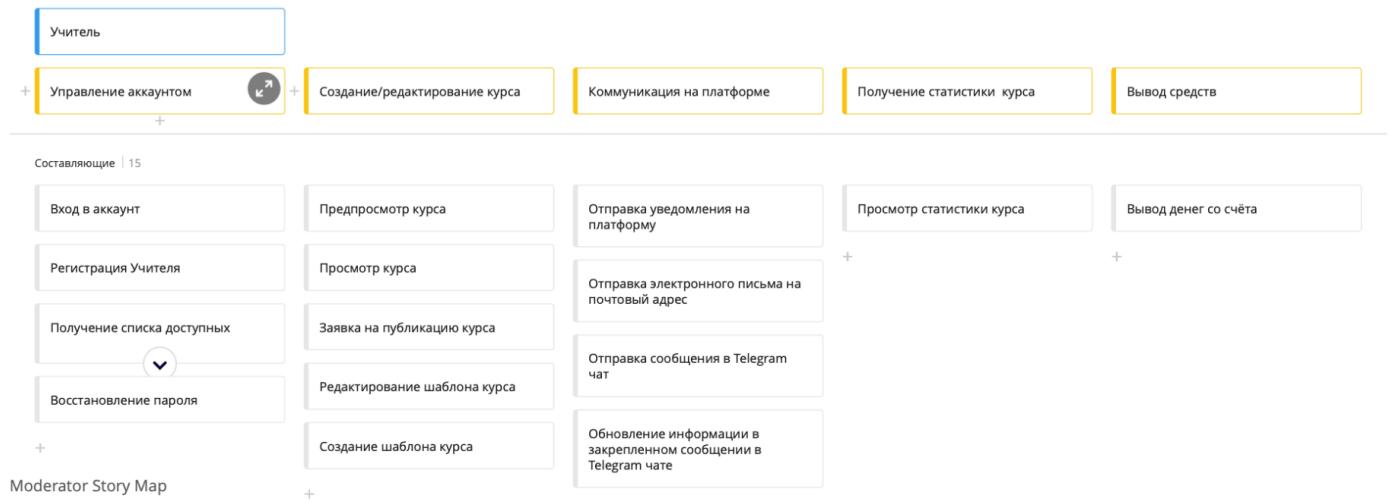
User Story Mapping

(модель относиться к Agile, но было разрешено и их добавлять)

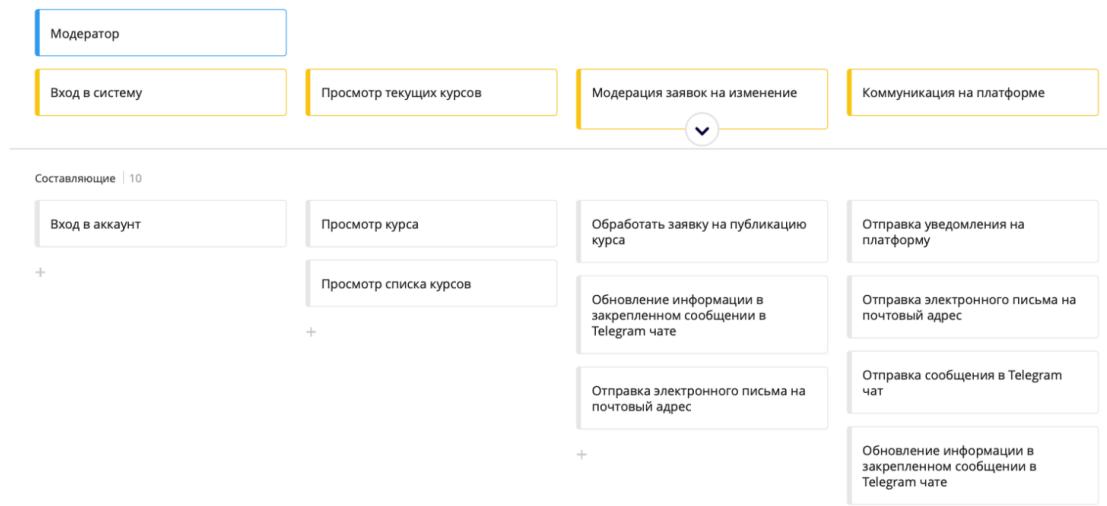
Student Story Map



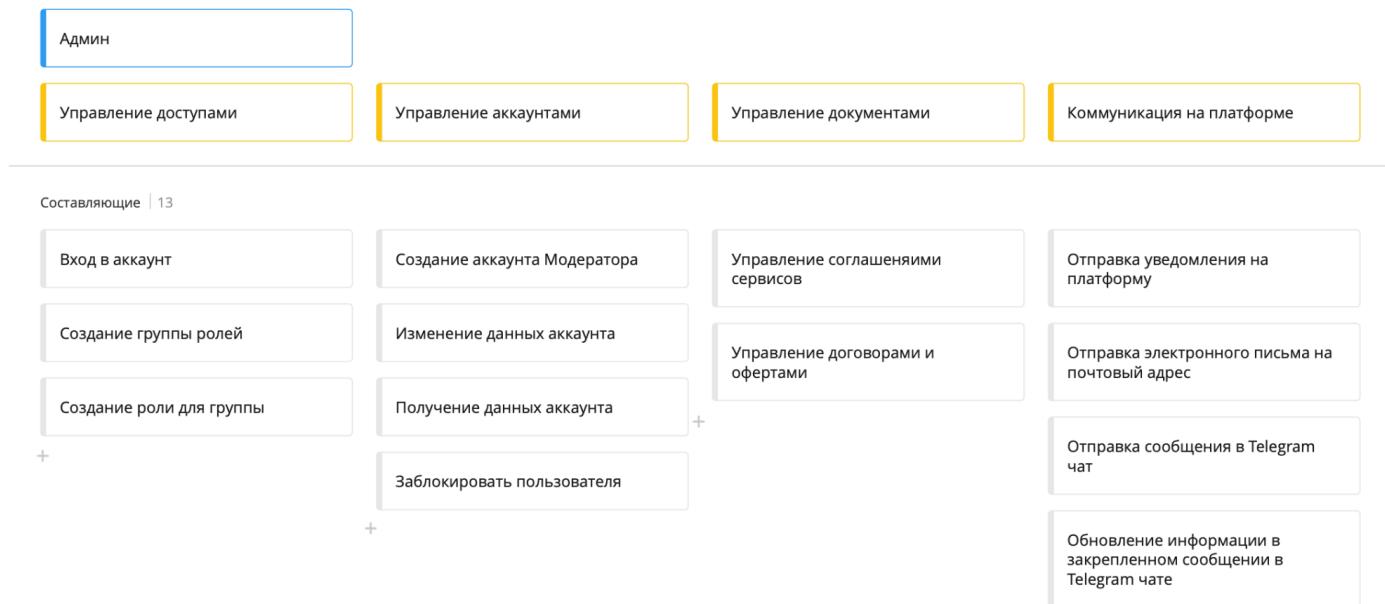
Teacher Story Map



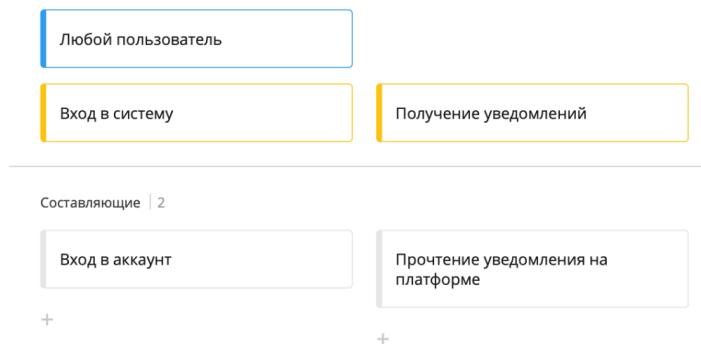
Moderator Story Map



Admin Story Map



User Story Map



Акторы данных кейсов написаны в столбцах сверху.

Системы реализующие кейсы приведены в таблице ниже для каждой функции.

Системы реализующие кейсы и разбиение на их функции подсистем:

приведены в таблице

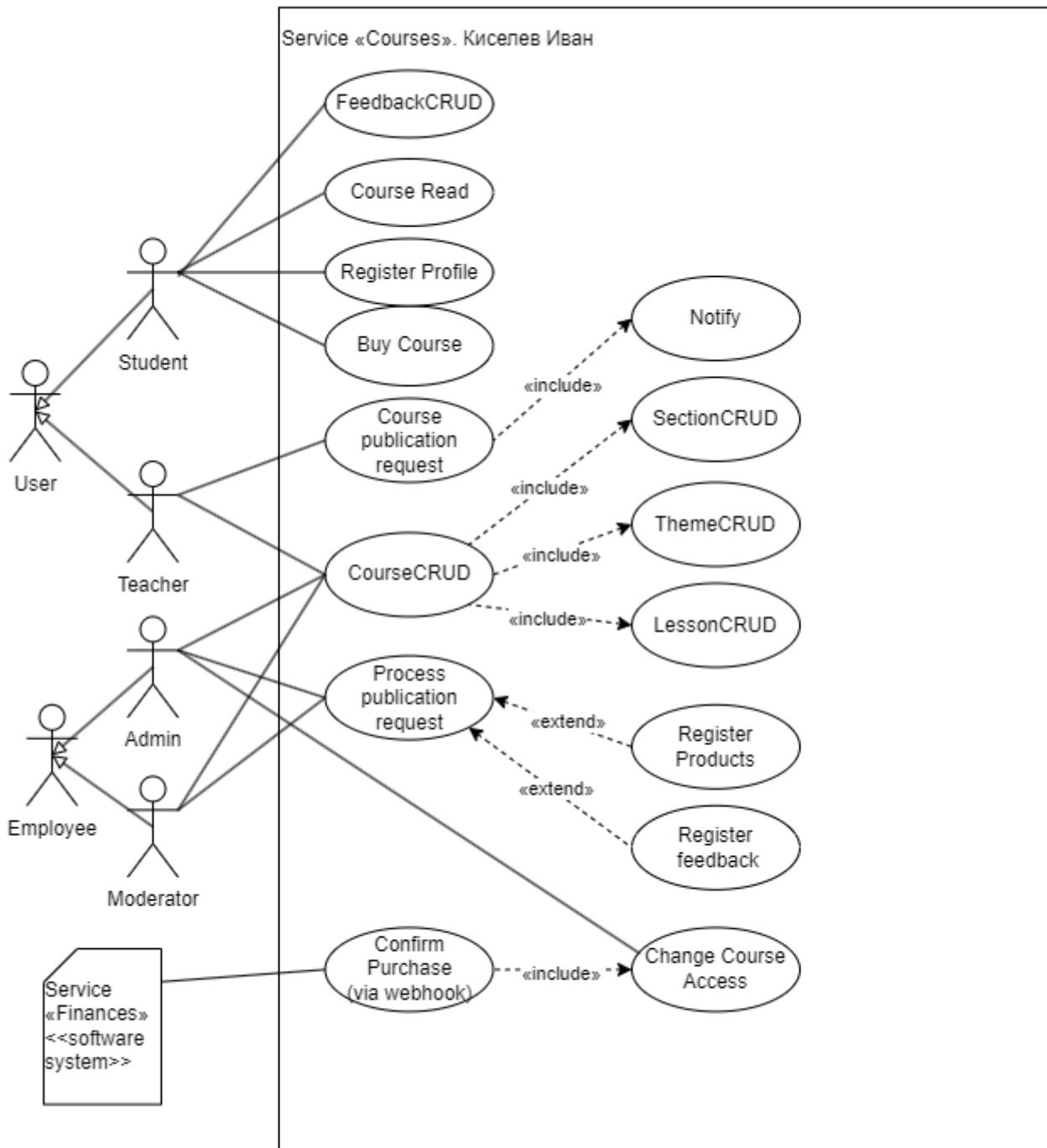
https://docs.google.com/spreadsheets/d/1BNtufvJ4I_MNSUXKp1r9HRXKvKay5HQm/edit?usp=sharing&ouid=109945936941129230421&rtpof=true&sd=true

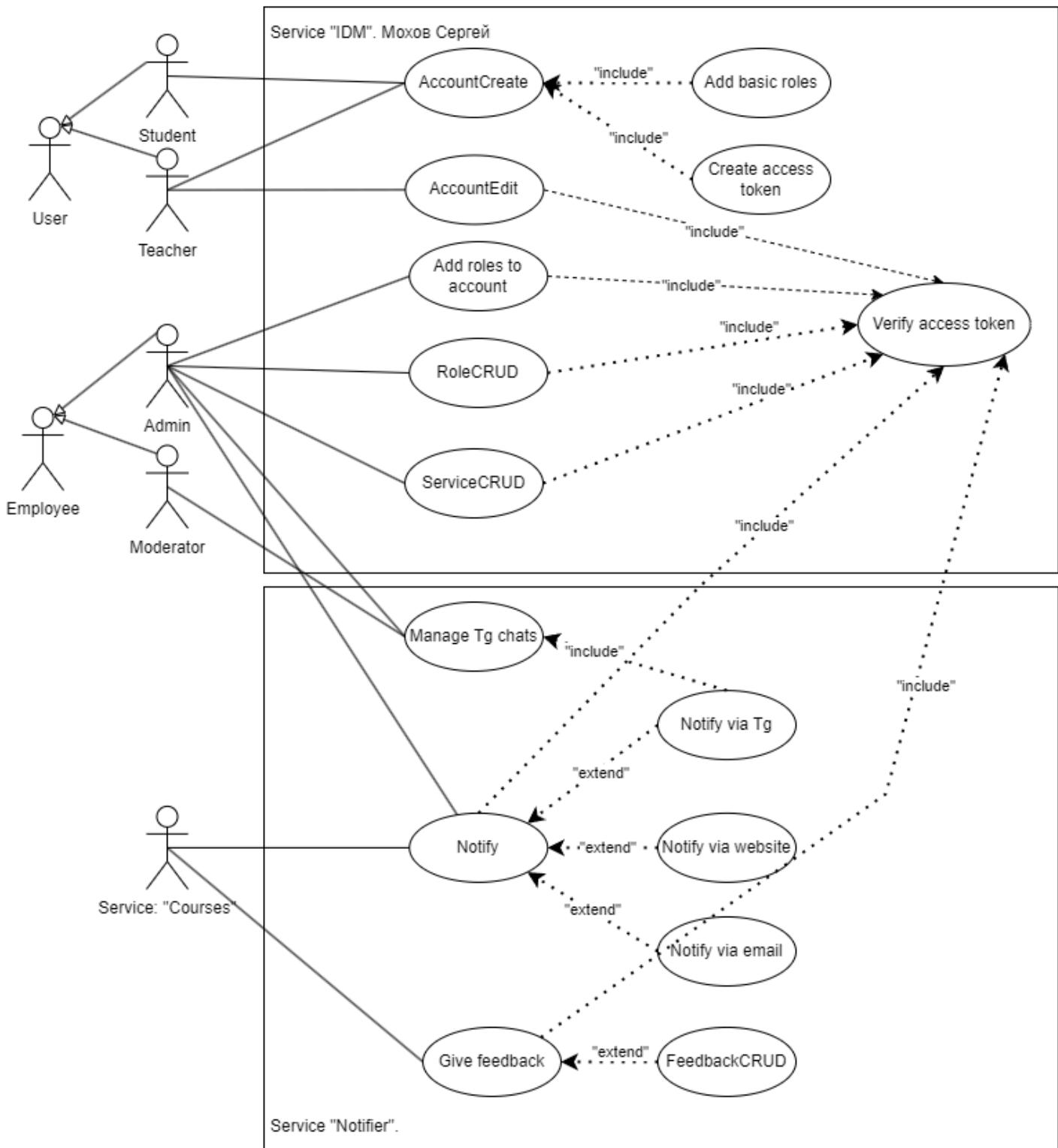
fucntion_map_platfrom.xlsx

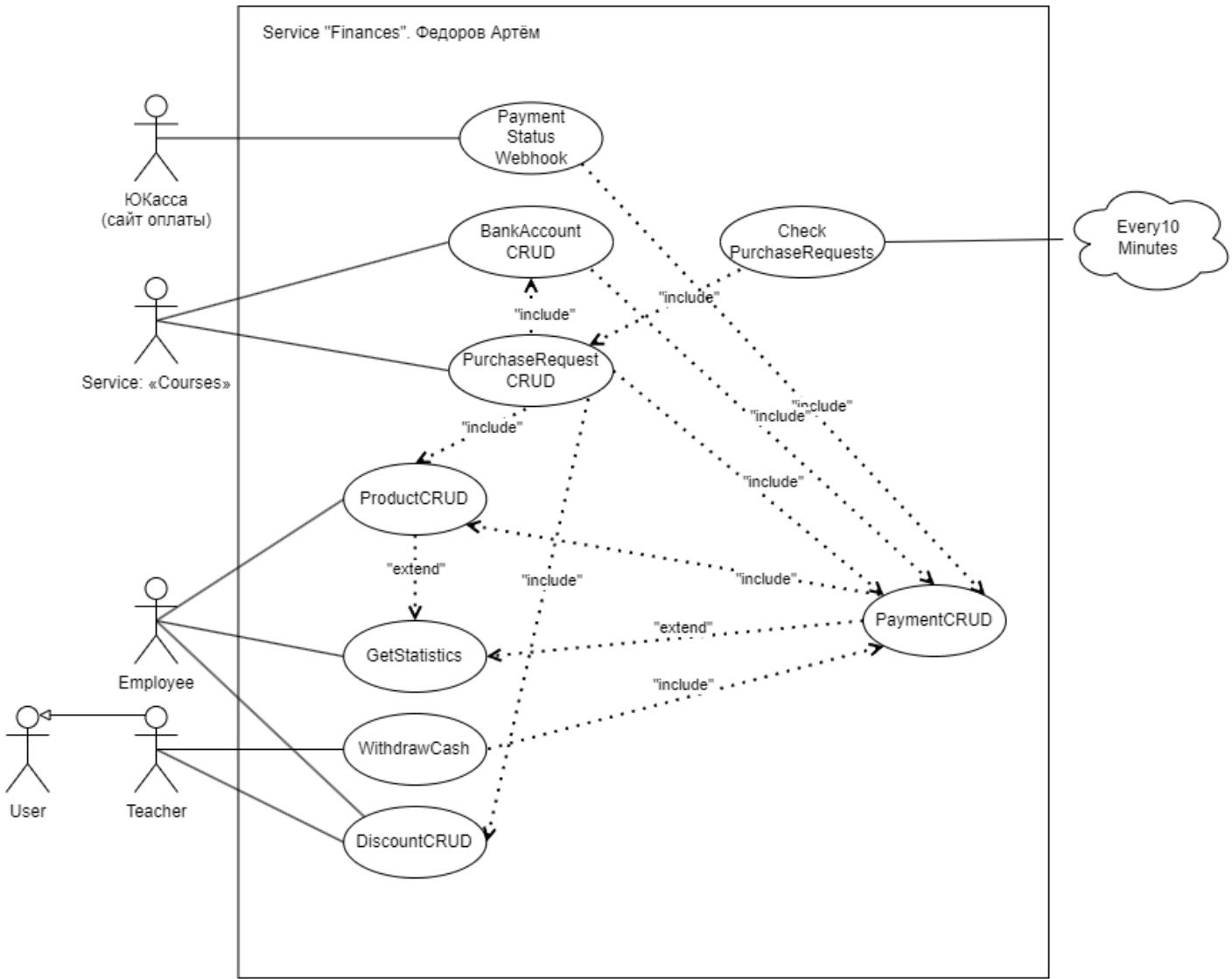
(Сюда не уместиться оригинал)

Use Cases (UML 2.5):

Для текущей итерации продукта были из данных бизнес кейсов сформированы use cases для подсистем (систем), подписанных сверху как Service.







Описание прецедента Course Read

Участники:

1. ПЛ - пользователь
2. CS – Course Service
3. БД – Database
4. FS – Finance Service
5. AS – Account Service
6. PS – Payment Service (Ю Касса)
7. БР - пользователь

(Service это система/подсистема)

1) Сценарий «Course Read»

Предусловие: ПЛ прошел этап авторизации (сценарий “Verify Access token”)

Описание: Данный сценарий описывает просмотр курса авторизованным пользователем

1. От ПЛ отправляется запрос в CS на конечную точку для выдачи курса
2. CS получает информацию о пользователе и искуому курсу из запроса
3. CS проверяет существование курса в БД по информации о нем | A1
4. CS достает соответствующий курс из БД
5. CS проверяет авторизован ли пользователь из информации приложенной в запрос из предусловия | A2
6. CS получает из БД список приобретенных пользователем частей курса

7. CS убирает материалы не приобретенных пользователем уроков из курса, игнорируя публичные (открытые всем пользователям) уроки
8. CS возвращает измененный курс пользователю

(A1) Альтернативный сценарий 1 “Данного курса нет”:

4. CS возвращает ошибку о том, что такого курса не существует

(A2) Альтернативный сценарий 2 “Пользователь не авторизован”:

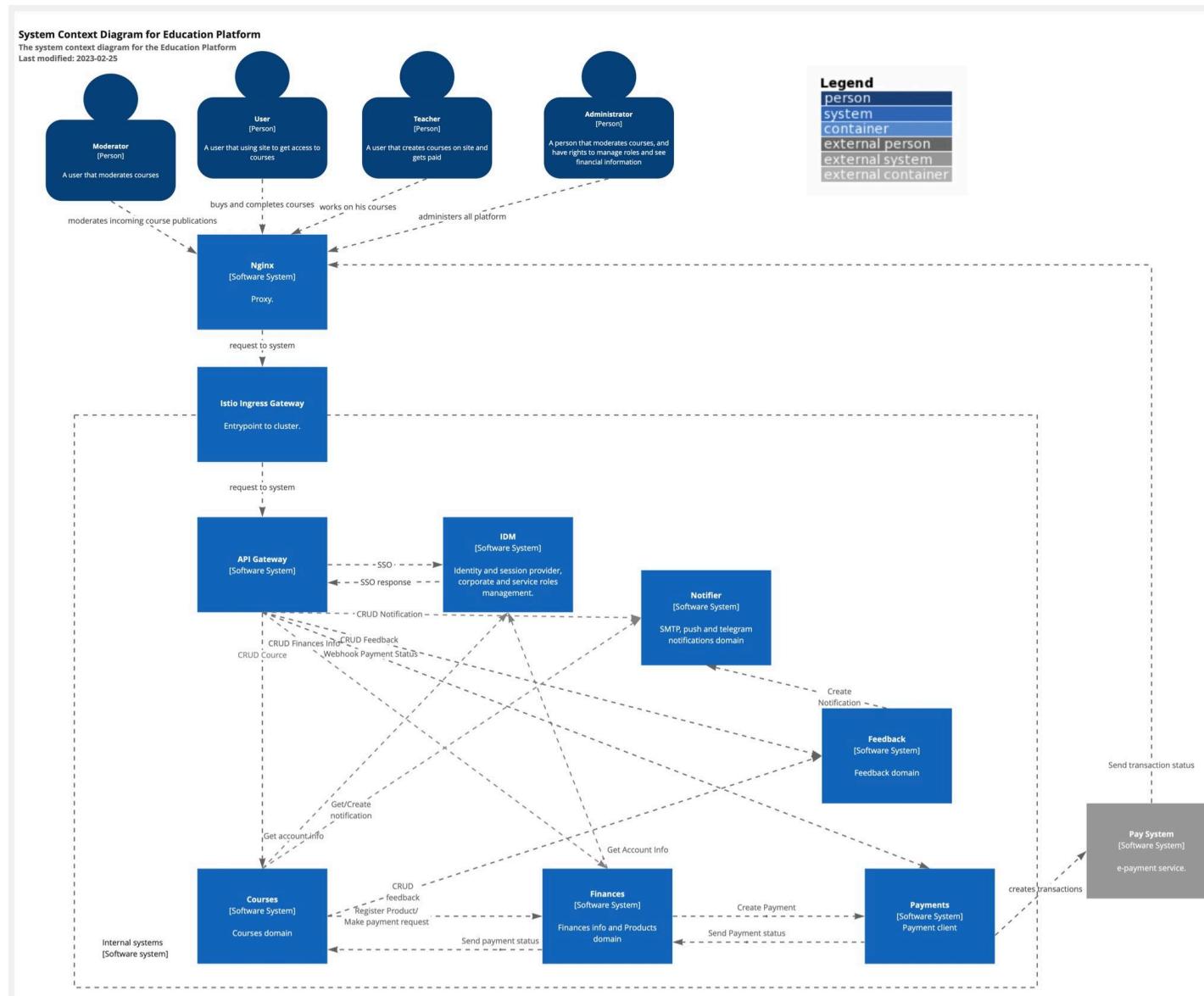
5. CS убирает материалы уроков из курса, игнорируя общедоступные уроки
6. CS убирает материалы уроков из курса, игнорируя общедоступные уроки
7. CS возвращает измененный курс пользователю

Про представление и модели

Данные модели и прецеденты показывают точки зрения акторов на их конечные цели и использование системы, выражая функциональный интерес. С точки зрения разработчиков данные схемы выражают закрепление исходных функций для автоматизации и разбиение функционала на подсистемы, что связано с их интересом к понятности и простоты в разработке системы. На основе этих прецедентов будут разрабатываться системы

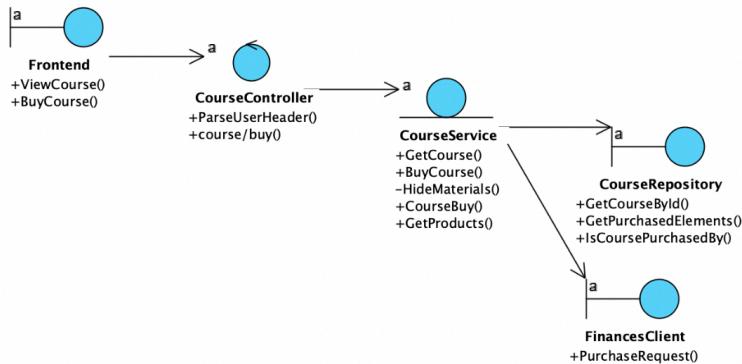
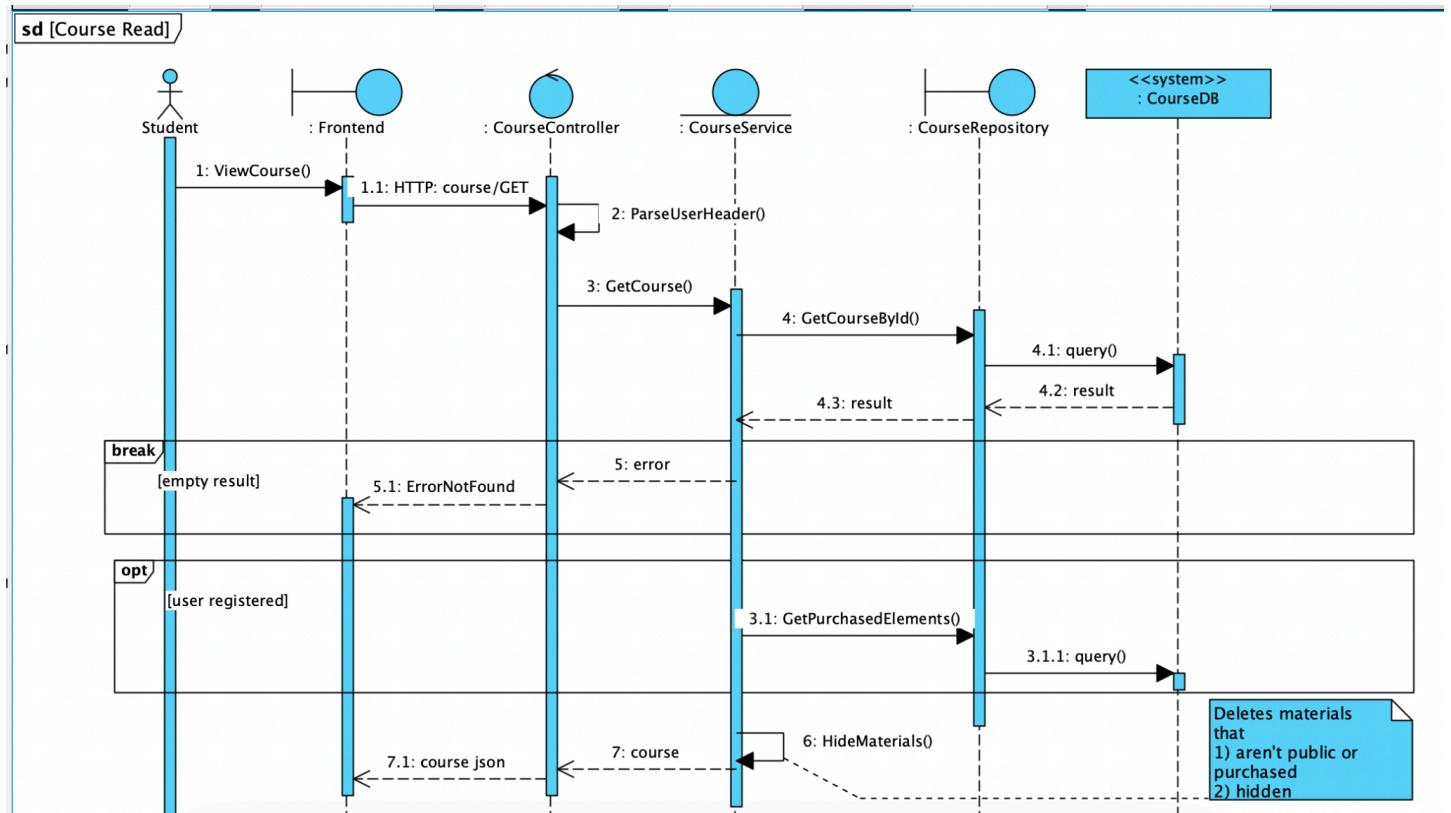
4.2. Логическое представление

C4 model Context level:



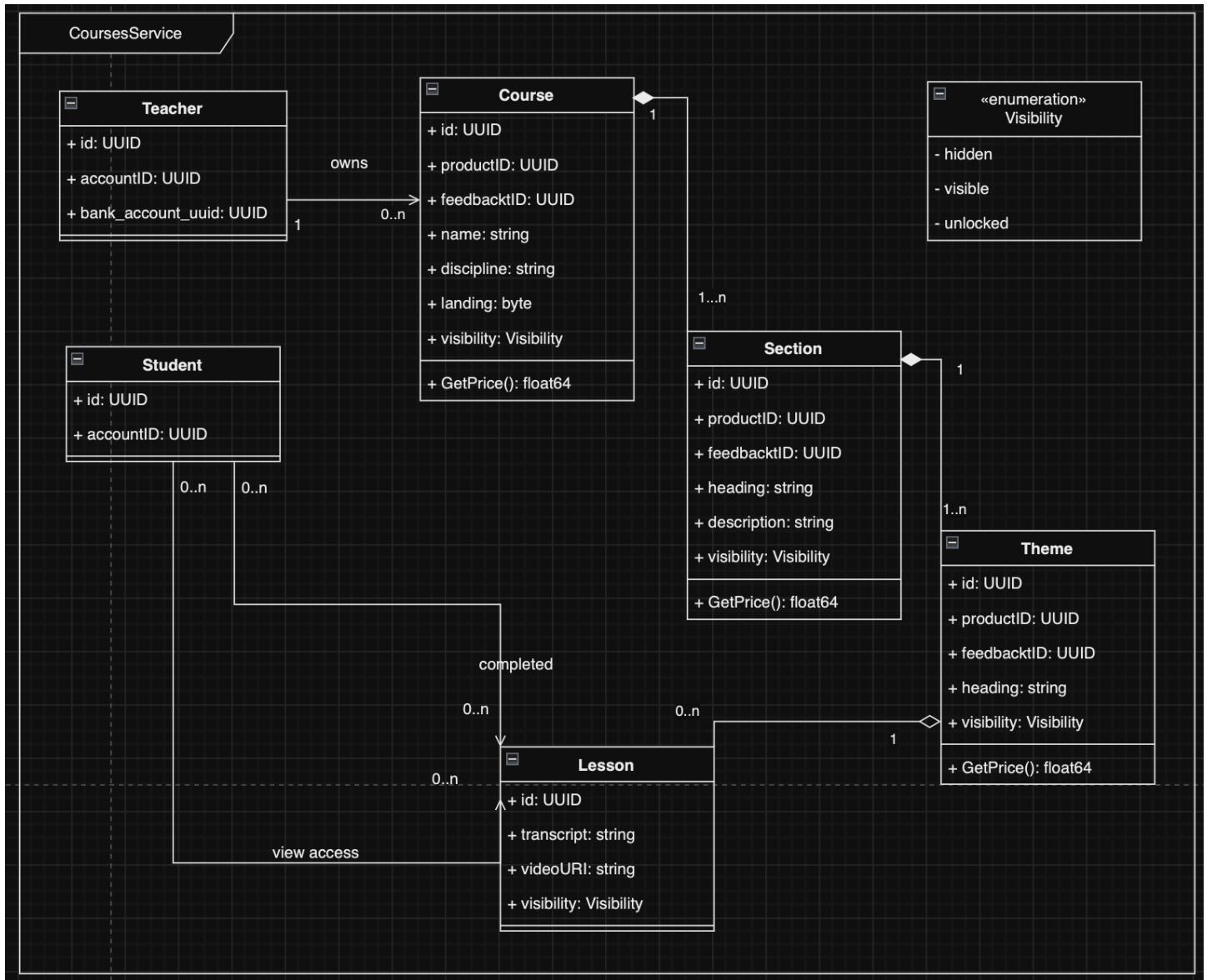
Данная модель представляет разбиение системы на компоненты-подсистемы и показывает связь между ними

Sequence diagram of "Course Read" Course system UseCase

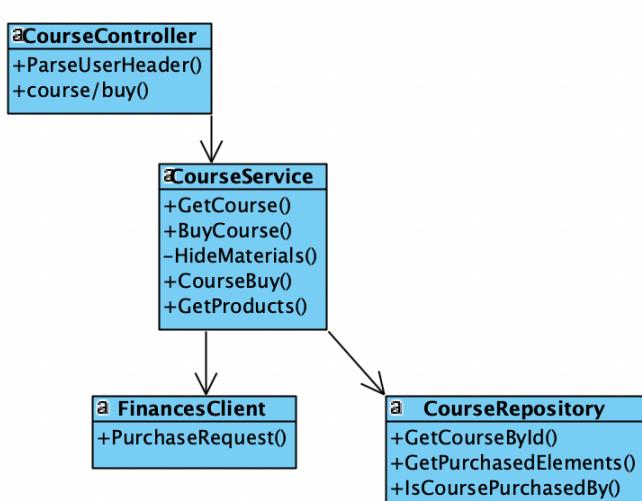


Данная диаграмма последовательность относится к интересу простоты разработки и сопровождения с точки зрения разработчика. Она показывает функциональный смысл разбиения программы на три слоя согласно чистой архитектуре.

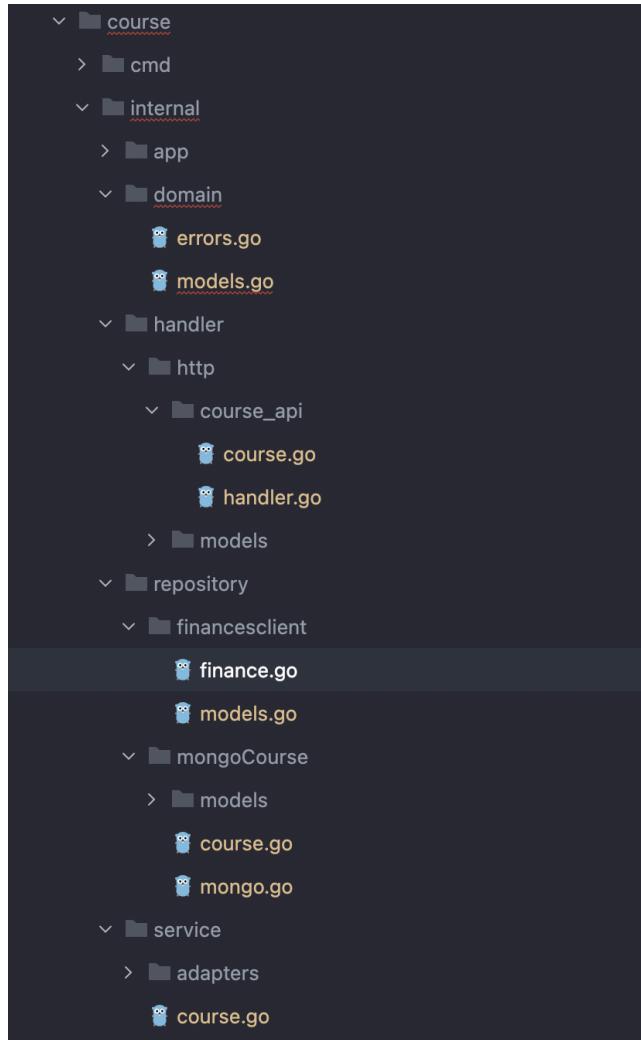
Course service class diagram:



Данные модели это “Domain models” будут расположены в одном пакете domain. Слой сервиса (бизнес логики) будет работать только с ними. Другие слои controller (транспорт) и repository будут парсить свои модели в эти domain модели. Ниже остальные классы реализующие и чистую архитектуру (стрелочка значит содержит ссылку на)



Разбиение на пакеты



4.3. Представление архитектуры процессов

В системе будут запущены следующие программы с диаграммы развертки,, между данными контейнерами будет асинхронное взаимодействие. Внутри программ паттерн контроллеры (реализованный с помощью HTTP роутера) будут запускать асинхронные процессы на каждый Use Case. Между данными процессами не будет состояний гонки.

4.4. Физическое представление архитектуры

Серверное оборудование:

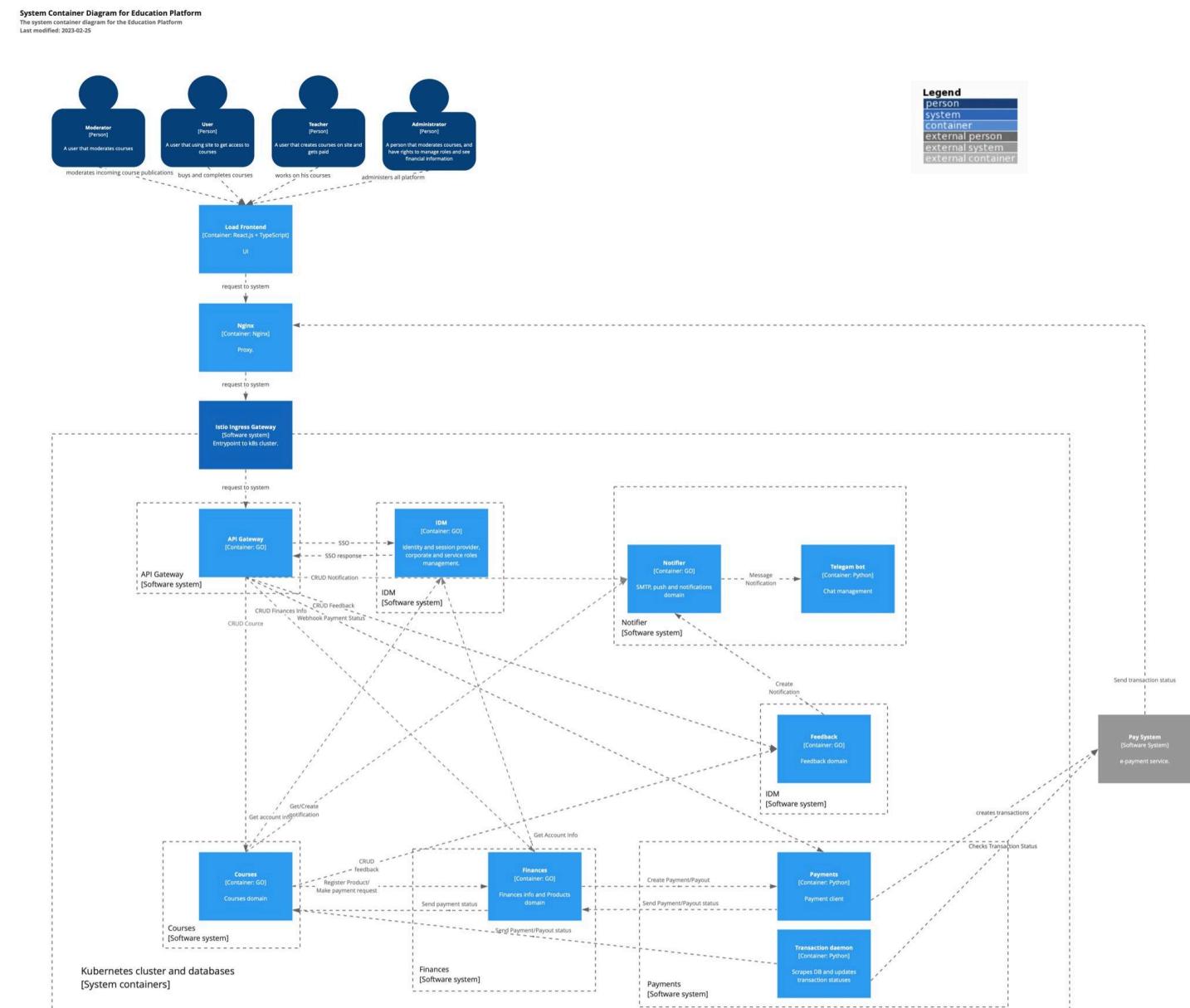
- Для хостинга веб-приложения и баз данных будет закуплен выделенный сервер.
- Сервер должен иметь достаточное количество вычислительных ресурсов (процессор, оперативная память) для обеспечения производительности и отказоустойчивости системы.
- Желательно использование серверов с поддержкой RAID-массивов для обеспечения безопасности данных и их резервного копирования.
- Для обеспечения хранения данных будет использоваться устройства сетевого хранилища
- Резервное оборудование должно быть готово к автоматическому восстановлению работы системы в случае сбоев или отказов основного оборудования.

Для систем из диаграммы развертывания ниже выделяется следующее железо, связи между системами должны поддерживать HTTP 2 и 3 общение

Системы	Оперативная память	Место на диске	Процессор	Доступ в Интернет
Gateway	4 ГБ	40 ГБ	4 Ядра	Нет
IDM	4 ГБ	40 ГБ	4 Ядра	Нет
Courses	4 ГБ	40 ГБ	4 Ядра	Нет
Finances	2 ГБ	40 ГБ	2 Ядра	Да
Notifier	2 ГБ	40 ГБ	2 Ядра	Да
Feedback	2 ГБ	40 ГБ	2 Ядра	Нет
Payments	2 ГБ	40 ГБ	2 Ядра	Да
Telegram Bot	2 ГБ	40 ГБ	2 Ядра	Да
Кластер PostgreSQL	16 ГБ	128 ГБ	4 Ядра	Нет
Кластер MongoDB	16 ГБ	128 ГБ	4 Ядра	Нет

4.5. Представление развертывания

Представление развертывания можно отлично видеть на модели Container уровня C4 model



Система будет обновляться вручную, новые версии из приватного репозитория на Github.

4.6. Представление архитектуры данных

Для сервиса courses будет создаваться replica set MongoDB. Гонки данных в этом сервисе не ожидается.

4.7. Представление архитектуры безопасности

Для защиты покупок сервиса courses:

пароли пользователей в IDM (SSO) будут хэшироваться алгоритмом Argon2 и добавлением 8 бит случайной соли на каждый пароль.

В js коде будет внутренняя проверка или обхождение XSS

Атаки на сеть будут решены с помощью изолирования сети и общений по сети, а также использования прокси и Ngix

Против CSRF атак мы не будем менять данные по POST запросам.

Все приватные данные будут отправляться в теле запроса.

4.8. Представление реализации и разработки

Команда пользуется репозиторием на Гит, собственными соглашениями об оформлении коммитов и pull requests, а также backlog составленным по User Story Mapping

4.9. Представление производительности

Будет использоваться JWT токены с SHA256 и подписью, а также алгоритмы L4 и L7 балансировки трафика в k8s.

4.10. Атрибуты качества системы

За счет инструментов k8s и репликации MongoDB планируется целевое значение доступности сервиса Courses не менее 99% времени.

Проверка безопасности будет производиться экспертизой.

4.10.1. Объем данных и производительность системы

Система должна обрабатывать данные о пользователях, курсах, финансах, уведомлениях и отзывах. Ожидается, что объем данных будет увеличиваться с увеличением числа пользователей и курсов. Производительность системы должна быть достаточной для обеспечения отзывчивости интерфейса и обработки большого количества запросов.

Для поддержки данной нагрузки будет использован kubernetes.

Критерии не выдвигаются в рамках данной разработки

4.10.2. Гарантии качества работы системы

Работоспособность системы будет гарантироваться регулярными тестами:

1. Unit-тесты для отдельных компонентов.
2. Основные интеграционные тесты для проверки взаимодействия между компонентами.
3. Некоторые End-to-end тесты для проверки работы системы в целом.
4. Тестирование безопасности для обнаружения уязвимостей и защиты от атак.

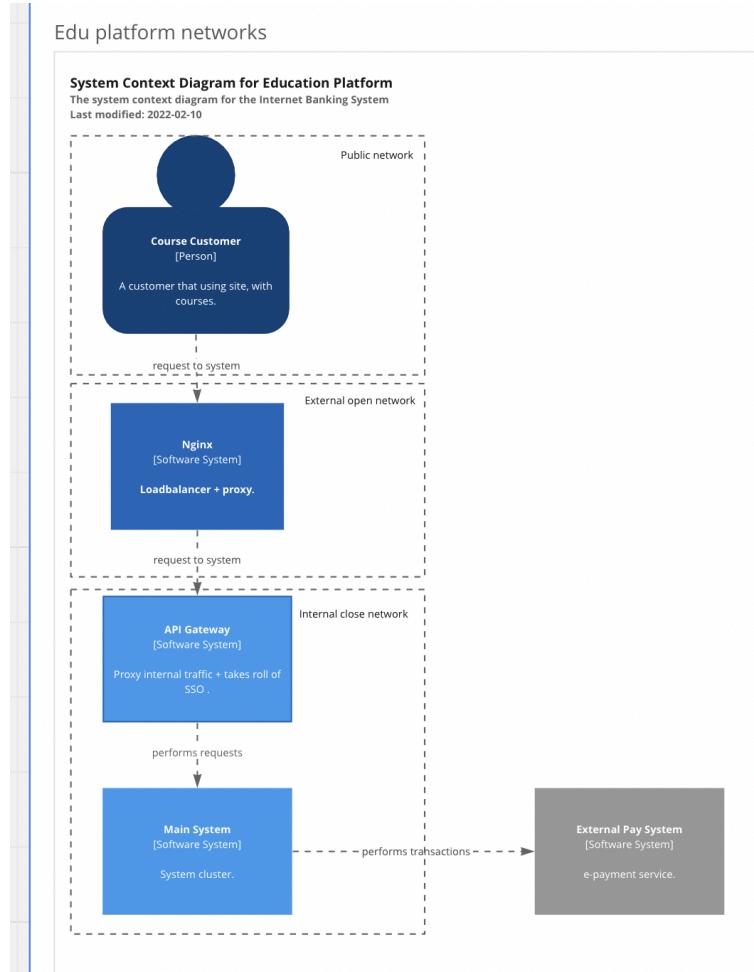
Эти тесты будут включены в процесс непрерывной интеграции и непрерывной поставки для обеспечения стабильной работы системы.

Планируется покрытие тестами 70-90%

На общие библиотеки в /pkg должны быть написаны unit тесты с покрытием 95-100%

Проверка будет идти согласно приемки по ТЗ

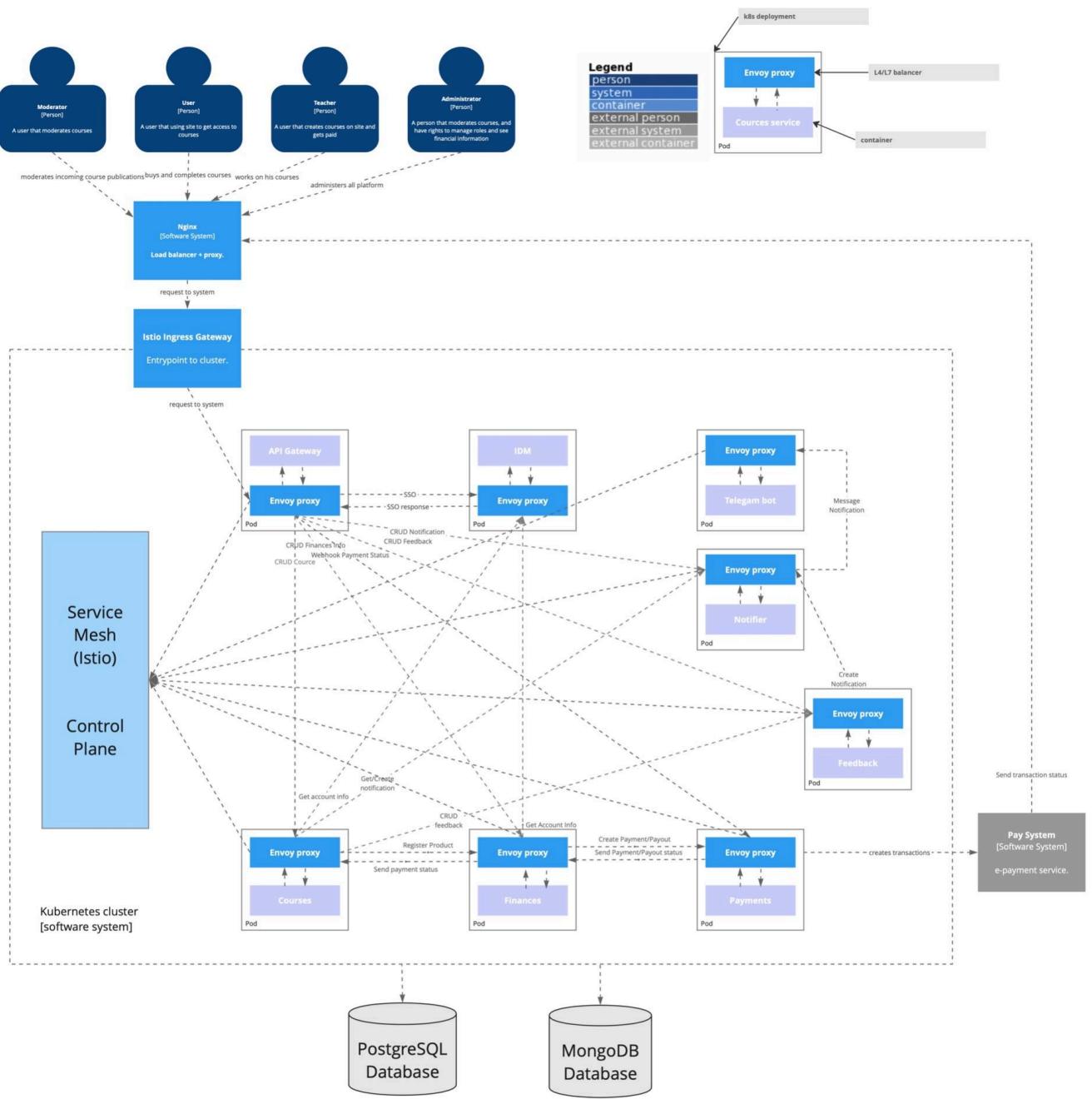
4.11. Представление сетей



Показывает как система будет защищаться от атак на сеть.

4.12. Представление использования Kubernetes

System k8s Container Diagram for Education Platform
The system context diagram for the Internet Banking System
Last modified: 2022-02-10



5. Технические описания отдельных ключевых архитектурных решений

5.1. Техническое описание решения использования микросервисной архитектуры

5.1.1. Проблема

Какой привычный подход к архитектуре приложения выбрать для интересов разработчиков: обеспечения масштабируемости, гибкости и независимости развития?

5.1.2. Идея решения

Использование микросервисной архитектуры. Разделение по DDD и функциям системы на асинхронные независимые системы микросервисы.

5.1.3. Факторы

- Требования к масштабируемости: Микросервисы обеспечивают горизонтальное масштабирование компонентов системы.
- Гибкость и независимость: Микросервисы позволяют разрабатывать, развертывать и масштабировать каждую подсистему независимо друг от друга.
- Возможность поддерживать разные команды и языки: Разделение системы на микросервисы дает возможность разрабатывать на разных языках в независимых командах.

5.1.4. Решение

- Разработка каждого компонента системы в виде отдельного микросервиса.
- Использование протоколов взаимодействия между микросервисами, таких как HTTP REST API или сообщения через брокера сообщений.

5.1.5. Мотивировка

Микросервисная архитектура обеспечивает гибкость, масштабируемость и независимость компонентов системы. Это позволит эффективно организовать разработку на разных языках и в независимых командах. Это также позволит горизонтально масштабировать системы и внедрять новые бизнес домены.

5.1.6. Неразрешенные вопросы

Нет.

5.1.7. Альтернативы

Альтернативным подходом к архитектуре может быть монолитное приложение, где все компоненты системы объединены в одном приложении. Данный подход гораздо дешевле, однако не подходит для дальнейшего расширения платформ.

Также альтернативы использующие события или шины сообщений для взаимодействия между сервисами и сбору ответу – не выбрались по причине своей сложности для текущей команды.

5.2. Техническое описание решения использование языка программирования Go

5.2.1. Проблема

Какой язык программирования выбрать для разработки требовательных к многопоточной нагрузке систем и их сопровождения ?

5.2.2. Идея решения

Использование популярного языка программирования Go для разработки.

5.2.3. Факторы

- Производительность, конкурентность и параллелизм: в контексте систем реализующих API Go имеет огромное преимущество засчет своей реализации работы с многопоточностью. Go может обрабатывать одновременно очень много асинхронных процессов – потоков в виде горутин.
- Простота разработки и сопровождения: Простой и лаконичный синтаксис, а также широкий стандартный набор библиотек делают разработку и сопровождение приложений на Go более простыми.
- Преимущества от стандартизации: Лаконичный синтаксис и реализация принципов, а также широкий стандартный набор библиотек делают разработку и сопровождение приложений на Go более стандартизируемым.

5.2.4. Решение

Использование языка программирования Go для написания подсистем, таких как Identity Manager, Courses, Finances, Notifier, Feedback и Gateway.

5.2.5. Мотивировка

Выбор Go обоснован его высокой производительностью в наших потребностях – горутины отлично эффективно и быстро работают благодаря уникальной модели многопоточности – M:N (или же M:P:N если учитывать внутренние логические процессоры). Выстроенное распределение работы M потоков на физических N процессорах, ядер с хорошими алгоритмами work stealing позволяет приложениям на Go не тормозить даже на железе с малым количеством ядер, что очень актуально для микросервисов. Эта модель не будет хорошо работать у других языков программирования, если ее добавят, потому что они в отличие от Go не реализовали такое быструю совместимость с языком C, на котором работают вызовы операционной системы для управления процессорами в go-runtime (реализующем управление разделения работы).

Go также легко сопровождать, если писать достаточно качественный код согласно выбранным общим архитектурным решениям и принципам

5.2.6. Неразрешенные вопросы

Нет.

5.2.7. Альтернативы

Учитывая важность технического фактора данная модель M:N многопоточности есть только у Erlang, но он не подходит по другим факторам, например, сопровождению.

5.3. Техническое описание решения применения принципов Чистой архитектуры в микросервисах на Go

5.3.1. Проблема

Как организовать внутреннюю структуру микросервисов на языке Go для обеспечения гибкости и расширяемости?

5.3.2. Идея решения

Применение принципов Чистой архитектуры в разработке микросервисов на языке Go.

5.3.3. Факторы

- Стандартизация: Чистая архитектура предлагает возможности одинакового по смыслу разбиения на компоненты в разных микросервисах
- Разделение ответственостей: Чистая архитектура предлагает разделение системы на независимые слои и домены. Это позволяет изолировать различные компоненты системы и уменьшить зависимости.
- Тестируемость: Чистая архитектура способствует написанию модульных и автономных тестов для каждого компонента системы, что повышает качество и надежность кода.

5.3.4. Решение

- Использование слоев: Разделение системы на уровни (транспорт, бизнес-логика, хранилище) для изоляции различных аспектов функциональности.
- Деление на большие домены по чистой, при этом тогда в сервисе будет почти всегда только один.
- Инверсия зависимостей: Использование интерфейсов для определения контрактов между компонентами, что позволяет легко заменять реализации и тестировать компоненты независимо друг от друга.

5.3.5. Мотивировка

Применение Чистой архитектуры в микросервисах на языке Go позволяет создать гибкие, поддерживаемые и расширяемые компоненты системы, которые можно одинаково проверять по качеству кода и стандартизировать. Более того разделение на такие независимые домены будет легко тестировать.

5.3.6. Неразрешенные вопросы

Нет.

5.3.7. Альтернативы

В качестве альтернативы можно рассмотреть использование других архитектурных подходов, таких как DDD (Domain-Driven Design) или Hexagonal Architecture. Однако, применение Чистой архитектуры часто является более подробным и принятым выбором для микросервисных систем на Go, в рамках стандартизации и написания тестов и Чистого кода.

