Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления
Кафедра интеллектуальных информационных технологий

К защите допустить:

Заведующий кафедрой ИИТ

_____ Д. В. Шункевич

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему:

ИНТЕЛЛЕКТУАЛЬНАЯ РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА ПО ФИЛОСОФИИ

БГУИР ДП 1-40 03 01 001 ПЗ

Студент	И. Д. Титлов
Руководитель	О. Г. Мятликова
Консультанты:	
от кафедры ИИТ	С. А. Самодумкин
по экономической части	Т. Л. Слюсарь
Нормоконтролёр	С. А. Самодумкин

Рецензент

Минск 2025

Решением рабочей комиссии
допущен(а) к защите дипломного проекта

Председатель рабочей комиссии

_____ Д. В. Шункевич
(Подпись) (Инициалы и Фамилия)

____ _____ 2025 г.

Учреждение образования
Белорусский государственный университет
информатики и радиоэлектроники

Кафедра интеллектуальных информационных технологий

УТВЕРЖДАЮ

Заведующий кафедрой ИИТ

_____ Д. В. Шункевич

___ _____ 2025 г.

ЗАДАНИЕ
на дипломный проект

Обучающемуся Титлов Иван Дмитриевич
(фамилия, собственное имя, отчество (если таковое имеется))

Курс 4 Учебная группа 121703

Специальность 1-40 03 01 Искусственный интеллект

Тема дипломного проекта Интеллектуальная рекомендательная система по философии
(наименование темы)

Утверждена руководителем УВО БГУИР №632-с от 25 марта 2025 г.

Исходные данные к дипломному проекту: Базы данных PostgreSQL; Средство контейнеризации приложении Docker; Языки разработки C#, React + TS, Python.

Перечень подлежащих разработке вопросов или краткое содержание расчетно-пояснительной записки:
Введение
1 Анализ подходов к разработке интеллектуальных приложений в предметной области Философия
2 Проектирование модели рекомендательной системы по философии
3 Разработка интеллектуальной рекомендательной системы по философии
4 Экономическое обоснование разработки программного обеспечения
Заключение

Перечень графического материала (с точным указанием обязательных чертежей и графиков):
Схема алгоритма согласованного управления оборудованием — формат А4, лист 1.
— формат А4, лист 1.
— формат А4, лист 1.
— формат А4, лист 1.

Консультанты по дипломному проекту (с указанием разделов, по которым они консультируют):
Раздел 1–3 — консультант от предприятия О.Г. Мятликова
Раздел 4 — консультант по экономической части Т.С. Слюсарь

Примерный календарный график выполнения дипломного проекта

№ п/п	Наименование этапа дипломного проекта	Объем этапа, %	Срок выполнения этапа
1	Обзор литературных источников по теме, изучение проблемной области	10	25.03 – 21.04
3	Определение требований к реализации	10	30.03 – 20.04
4	Проектирование модели подсистемы	25	10.04 – 30.04
5	Разработка подсистемы	30	21.04 – 20.05
6	Экономическое обоснование принятого решения, определение экономической эффективности внедрения полученных результатов	10	01.05 – 20.05
7	Оформление расчетно-пояснительной записки	10	30.03 – 29.05
8	Оформление графического материала	5	01.05 – 29.05

Дата выдачи задания 25 марта 2025 г.
Срок сдачи студентом законченного дипломного проекта 30 мая 2025 г.

Руководитель дипломного проекта _____ О. Г. Мятликова
(подпись) (инициалы, фамилия)

Подпись обучающегося _____

Дата ____ _____ 2025 г.

РЕФЕРАТ

ИНТЕЛЛЕКТУАЛЬНАЯ РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА ПО ФИЛОСОФИИ: дипломный проект/ И. Д. Титлов. – Минск : БГУИР, 2025, – п.з. – 68 с., чертежей (плакатов) – 4 л. формата А1.

Целью дипломного проекта является разработка рекомендательной системы по философии.

Предметом исследования являются подходы к разработке рекомендательных систем, самые современные и удачные архитектурные решения при их реализации в приложениях.

Данный программный модуль позволяет осуществлять поиск по предметной области философия, получая необходимые рекомендации.

В первом разделе пояснительной записки проведён анализ подходов к разработке рекомендательных систем, различных типов рекомендательных систем.

Во втором разделе выполнено проектирование рекомендательной системы по предметной области «Философия».

В третьем разделе описана разработка приложения, реализующего данную рекомендательную систему.

В четвертом разделе приведено технико-экономическое обоснование разрабатываемого программного продукта.

Результатом дипломного проектирования является программа «Философский каталог», содержащая сервер, сервис для построения рекомендаций, клиент и базу данных, которые реализуют поставленные в пояснительной записке задачи.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

API (от англ. Application Programming Interface) — программный интерфейс приложения;

BPMN (от англ. Business Process Modeling Notation) — диаграмма бизнес-процессов;

CI/CD (от англ. Continuous Integration/Continuous Deployment) — непрерывная интеграция/непрерывное развертывание;

CLS (от англ. Classify token) — токен классификации;

GIN (от англ. Generalized Inverted Index) — обобщённый обратный индекс;

GIST (от англ. Generalized Search Tree) — обобщённое поисковое дерево;

GUI (от англ. Graphical User Interface) — графический пользовательский интерфейс;

JSON (от англ. JavaScript Object Notation) — Нотация объектов JavaScript;

LLM (от англ. large language model) — большая языковая модель;

MVP (от англ. Model-View-Presenter) — Модель-Представление-Презентер;

MVVM (от англ. Model-View-ViewModel) — Модель-Представление-ViewModel;

MVC (от англ. Model-View-Controller) — Модель-Представление-Контроллер;

ORM (от англ. Object-Relational Mapping) — объектно-реляционное преобразование;

TF-IDF (от англ. TF — term frequency, IDF — inverse document frequency) — частота термина, обратная частота документа;

SQL (от англ. Structured Query Language) — язык структурированных запросов.

ВВЕДЕНИЕ

В современных коммерческих приложениях положительный пользовательский опыт является одной из главных целей разработчиков, именно он заставляет пользователя выбирать приложение среди всего массива документов и становится его постоянным посетителем. Одним из важных критериев получения этого опыта — индивидуализация контента для пользователя, в таком случае не просто упрощается достижение целей, ради которых человек приходит в приложение, но и происходит попеременное увлечение продуктом, когда просмотр одной сущности сразу по цепочке вовлекает к следующей. Один из основных способов достижения такой — индивидуализированная рекомендательная система, предлагающая пользователю индивидуальный опыт взаимодействия с продуктом.

На данном этапе практически каждое крупное приложение обладает рекомендательной системой: интернет-магазины, стриминговые сервисы, видеохостинги и системы аренды жилья. При всём этом многообразии решений есть предметные области трудно поддающиеся коммерциализации, а значит куда реже встречаются рекомендательные системы по научным и интеллектуальным областям. Данный дипломный проект призван заполнить пробел в имеющихся продуктах и предоставить пользователям бесплатный доступ к рекомендательной системе по философии.

Пояснительная записка содержит анализ, посвященный изучению решению в крупных коммерческих проектах и основных алгоритмов и технологий реализаций этих решений. В проектировании будет разобрана модель архитектуры, наиболее подходящая приложению и построены подробные диаграммы, по котором будет осуществляться дальнейшая разработка приложения. Наконец, разработка будет посвящена инструментам реализации, а также будет содержать листинг кода и скриншоты реализованного приложения.

Дипломный проект выполнен самостоятельно, проверен в системе "Антиплагиат". Процент оригинальности соответствует норме, установленной кафедрой ИИТ. Цитирования обозначены ссылками на публикации, указанные в "Списке использованных источников".

1 АНАЛИЗ ПОДХОДОВ К РАЗРАБОТКЕ РЕКОМЕНДАТЕЛЬНЫХ СИСТЕМ

В рамках практики были изучены множество подходов к такой фундаментальной в современных клиентоориентированных решениях вещи, как система рекомендаций, которая позволяет персонализировать и индивидуализировать опыт пользователя в приложении. На данный момент ни одно крупное коммерческое приложение не обходится без той или иной системы рекомендаций, разработанной одним из «классических» или более современных способов. Вначале определим подходы к реализации, которые определяют, каким образом пользователю формируются те или иные рекомендации. Базовый алгоритм изображен на рисунке 1.1:

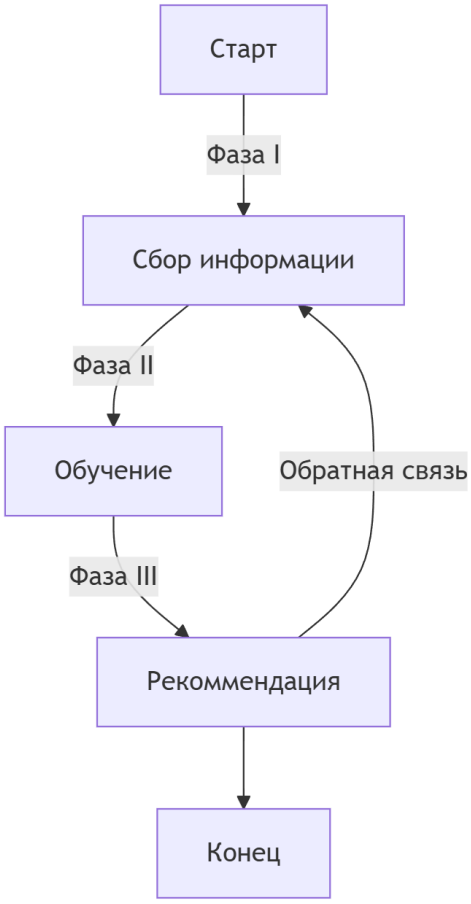


Рисунок 1.1 – Базовые шаги по формированию рекомендаций в рекомендательной системе.

1.1 Анализ подходов к построению рекомендательных систем

Существует несколько основных подходов к построению таких систем: коллаборативная фильтрация, контентно-ориентированная фильтрация и гибридные методы.

1.1.1 Коллаборативная фильтрация

Данный подход к рекомендациям построен на взаимодействии пользователей и предполагает, что пользователи, имеющие похожие предпочтения в прошлом, будут иметь те же предпочтения и в будущем. Исходя из этого, для подбора рекомендаций пользователю анализируется активность пользователей с похожей историей просмотренного и понравившегося контента. Данная система рекомендаций делится на два типа [1]:

– **Методы, основанные на памяти (memory-based)**, более простые, «классические» методы, которые используют информацию о предпочтениях пользователей напрямую для формирования рекомендаций, применяя алгоритмы, такие как k-ближайших соседей и корреляция Пирсона и находя наиболее соответствующие вам группы по интересам среди пользователей данного приложения.

– **Методы, основанные на модели (model-based)** в свою очередь включают построение моделей на основе данных о предпочтениях пользователей с использованием методов машинного обучения, таких как матричная факторизация и вероятностные графические модели. Данный подход учитывает большее число факторов и обеспечивает большую точность рекомендаций, в последнее время с появлениям эффективных больших языковых (далее — LLM) моделей они начали преобразовываться в *методы с использованием глубокого обучения.*

– **Методы с использованием глубокого обучения** — методы, получившие распространение лишь совсем недавно во время резкого прогресса в развитии LLM-моделей, предполагает именно их использование для коллаборативной фильтрации, однако на данный момент действительная эффективность LLM-моделей по сравнению со стандартными методами предоставления рекомендаций остается под вопросом [2].

Данная система предполагает, что если у двух пользователей было одинаковое мнение насчёт некоторого контента, то оно будет с большей вероятностью одинаковым и насчёт остального контента. Исходя из этого, строятся рекомендации.

При этом при использовании коллаборативной фильтрации мы должны обращать внимание на несколько достаточно важных проблем её реализации: проблему «холодного старта», масштабируемость и разреженность.

Первая проблема — *«холодный старт»* возникает с новоприбывшими пользователями, в данном случае отсутствие данных может помешать формировать рекомендации, в таком случае многие сайты, такие как, например, Spotify или TikTok, реализовывают так называемый алгоритм «многорукого бандита», в котором при старте пользователю даётся набор тем и ему предлагается выбрать интересующие его. При этом при выборе одной темы (группы, предметной области и так далее) появляется выбор

из ещё нескольких схожих, которые, как считается, могут заинтересовать пользователя.

Второй проблемой является *масштабируемость*, в крупных системах появляются миллионы продуктов и пользователей, в данных условиях проведение и хранение результатов постоянно меняющихся сложных расчётов ложится крупной нагрузкой и может снизить производительность приложения.

Третьей проблемой является потенциальная *разреженность*, с которой особенно часто встречаются онлайн-сайты розничной торговли. На таких сайтах широкий реестр товаров, в связи с которым у многих товаров или вовсе нет оценок, или же их слишком мало, чтобы формировать полноценную группу по предпочтениям [3].

1.1.2 Фильтрация на основе содержания

Также крайне распространённый подход к построению рекомендательной системы. В отличие от первого подхода, в данном случае предполагается, что информации о других пользователях крайне мало или она вовсе отсутствует, по крайней мере, она не требуется для формирования рекомендаций. Последние в данном случае формируются на основе известной информации о понравившейся пользователю сущности (такой, как, например, её название, предметная область, расположение и прочие характеристики). На основе пользовательских «лайков» и «дизлайков» собирается информация по предпочтениям пользователя, при этом новые сущности в выборке сравниваются со старыми, уже находившимися в ней до этого, в попытке сформировать наиболее точный результат. В итоге получается особый профиль пользователя, в котором, в основном в виде словаря, хранятся его модель предпочтений и история взаимодействия с рекомендательной системой. На основе собранной информации система старается подобрать сущность, наиболее соответствующую и похожую на те, которые понравились пользователю в прошлом, отметая варианты, похожие по характеристикам и не заинтересовавшие пользователя.

В данном случае построенная модель предпочтений должна выстроить точную систему весов, от которой и будет отталкиваться будущая система. Самый распространенный алгоритм для выстраивания подобной математической модели, часто представленной в виде графа, — **TF-IDF**, при котором рекомендательная система формирует вес определенной сущности прямо пропорционально её популярности у пользователя и обратно пропорционально её частоте встречаемости в базе знаний системы; последнее в данном случае позволяет минимизировать артефакты конечного результата, отсеивая промежуточные переходы и индивидуализируя конечный профиль пользователя. После формирования весов дальнейшие алгоритмы

рекомендательной системы варьируются от обычного нахождения средних значений веса вектора определённой сущности до более сложных подходов от кластерного анализа до дерева решений с нейронными сетями [4].

При этом одной из основных проблем, стоящих перед разработчиком, является возможность рекомендовать иные, отличные от тех, по которым сформирована модель, категории предметов. В случае, если это невозможно, польза такой рекомендательной системы становится куда более низкой. Для обеспечения данной возможности чаще всего прибегают к архитектуре гибридных рекомендательных систем, объединяющих данный подход с коллаборативной фильтрацией, а также нейросетевым обучением. Эта комбинация позволяет улучшить качество рекомендаций для пользователя. Более подробно данные подходы рассмотрены в подразделе ниже.

1.1.3 Гибридная фильтрация

Почти все современные коммерческие приложения, в том числе те, которые будут перечислены в анализе ниже, используют данный подход к фильтрации, объединяющий в себе алгоритмы и способы формирования рекомендательной модели упомянутых выше методов. Он может представлять собой формирование моделей на основе коллаборативной фильтрации и фильтрации на основе контента отдельно с их последующим объединением, так и применять решение с последовательным применением алгоритмов для коррекции результатов. В целом, решения можно объединить в следующий список способов:

– взвешенный подход, при нём происходит численное объединение конечных оценок различных компонентов рекомендаций;

– переключающий подход осуществляет выбор между компонентами рекомендаций и применение выбранного компонента;

– смешанный подход объединяет различные рекомендательные алгоритмы вместе для получения формирования конечной модели еще до получения конечных оценок по каждому из подходов;

– при каскадном подходе к гибридизации рекомендации имеют строгий приоритет, при этом рекомендации с более низким приоритетом несколько меняют конечные результаты рекомендаций с более высоким, однако в целом имеют меньшее влияние;

– при метауровневом подходе применяется один рекомендательный алгоритм и создается некая модель, которая затем используется следующим алгоритмом [5].

Каждый из этих подходов имеет свои плюсы и минусы, а также нюансы применения, которые нужно учитывать при разработке. Однако стараться объединить несколько алгоритмов для обеспечения большей гибкости и точности конечного результата является хорошей и очень распростра-

нённой стратегией в современных приложениях. Для полного понимания проанализированных на теоретическом уровне подходов стоит провести анализ некоторых рекомендательных систем в популярных коммерческих приложениях.

1.2 Анализ рекомендательных систем коммерческих приложений

В рамках анализа рассмотрим несколько популярных приложений, в том числе стриминговый сервис *Netflix*, интернет-магазин *Ozon* и видеохостинг *YouTube*, что поможет понять как вышеупомянутые паттерны, так и общий подход к их реализации в современных условиях и с современными вызовами, особенно с пониманием того, что учебники, формирующие теорию, часто не успевают за устоявшейся практикой.

1.2.1 Интернет-магазин Ozon

Приложение **Ozon** — крупный российский интернет-магазин разнообразных товаров, который концентрирует широкий ассортимент продукции в самых разнообразных и при этом слабосвязанных категориях, таких как электроника, продукты, косметика и книги. Хранит лидерство на рынке в том числе благодаря продуманной системе рекомендаций, основанной на гибридной фильтрации с преобладанием фильтрации на основе содержимого. Рекомендуемые товары отображаются пользователю в виде отдельного виджета на странице. Шаги формирования рекомендаций представляют собой следующее:

1. **Подготовка контекста**, в рамках которой собирается информация о пользователе и товаре, для которого подбираются рекомендации. Список товаров в контексте включает товары, с которыми взаимодействовал пользователь, и товары, на странице которых находится виджет, отображающий рекомендации. Для каждого товара собирается его информация, в том числе список категорий, бренд, склад и так далее. О пользователе также берется информация о его поле и возрасте. Далее рекомендательный сервис выбирает несколько тысяч товаров (согласно заявлениям компании, около 2-4 тысяч), которые релевантны собранному контексту.

2. Следующим шагом происходит **оценка моделями машинного обучения**, при оценке модель определяет вероятность взаимодействия с каждым товаром из отобранных. При этом учитываемыми типами взаимодействия являются как покупки товаров, так и добавление их в особый список «избранных». Алгоритм оценки делится на несколько этапов:

   1) Вычисляются и рассматриваются все свойства, которые могут повлиять на факт продажи, при этом каждое свойство преобразовано

в скалярное взвешенное значение, являющееся вещественным числом, примерами свойств могут быть рейтинг товара и число отзывов на нём.

2) К данному набору свойств применяются алгоритмы машинного обучения, основной из используемых алгоритмов — метод градиентного спуска. Данные алгоритмы предсказывают вероятность совершения взаимодействий на основе значений свойств, при этом учитывается более 300 различных уникальных свойств товара.

3. Далее происходит **финальное ранжирование**, во время которого каждому товару-кандидату из списка рекомендаций сопоставляется его оценка релевантности, которая в общем случае учитывает:

– оценки моделей машинного обучения на прошлом этапе;

– рекламную ставку, в случае если у товара есть рекламное продвижение;

– насколько давно была посещена страница товара;

– иные характеристики товара, такие как скорость доставки и наличие на складе.

4. Наконец, товары отображаются в виджете, отсортированные в зависимости от оценки релевантности: от наиболее высокой до наиболее низкой [6].

Как можно заметить, в данном случае преобладает оценка на основе содержимого с использованием технологий машинного обучения, что происходит во многом ради учета интересов продавцов и рекламодателей, а также необходимости четко маневрировать между сложной системой категорий. Элементы коллаборативной фильтрации ловко внедрены лишь как одни из свойств товара, которые учитываются рекомендательной системой.

1.2.2 Видеохостинг YouTube

Youtube — самый популярный видеохостинг в мире, который завоевал многомиллиардную аудиторию по всему земному шару. На данный момент «Ютуб» предлагает невероятно широкую палитру пользовательского контента от прохождений игр до обучающих видео. Один из столпов успеха данного видеохостинга — очень продуманная и качественная рекомендательная система, прошедшая проверку временем. В данный момент алгоритмы «Ютуб» строятся на комплексной и дорогостоящей модели данных, в которую собирается информация со всех приложений, находящихся под управлением многомиллиардной корпорации *Google*. При этом в самом «Ютубе» стержень всей системы лежит на фильтрации, основанной на свойствах содержимого, в том числе играют известную роль следующие характеристики взаимодействия:

– Клики: одним из основных показателей для рекомендательной системы «Ютуба», что ты заинтересован рекомендацией и что её можно

считать хотя бы частично удачной — твое нажатие и открытие видео;

– время просмотра: очевидно, что нажатие на видео не означает, что тебе стоит в дальнейшем рекомендовать подобные видео или видео с этого канала, со временем на ютуб появилось много так называемых «кликбейтных» видео, завлекающей обманчивой и кричащей обложкой, которая не соответствует содержанию видео. К примеру, пользователю может попасться видео, в названии которого заявлено, что это матч между двумя любимыми командами, однако на деле в этом видео лишь обсуждение этого матча автором канала, лежащим на диване. В таком случае пользователь не будет долго смотреть видео, а почти сразу перейдёт на другое, что уже означает, что его влияние на модель рекомендательной системы почти не будет учитываться. Долгое время просмотра, а лучше полностью просмотренное видео, показывает устойчивый интерес пользователя к видео;

– ответы на вопросы в опросах: часто «Ютуб» даёт пользователям краткие опросники из одного вопроса, ответ на который также учитывается при формировании будущих рекомендаций;

– лайки, дизлайки, нажатие кнопки «поделиться»: всё это является основным способом взаимодействия с контентом у пользователя, показывая его реакцию на увиденное, что, очевидно, также учитывается в модели.

При этом стоит отметить, что при своём основании в 2008 году рекомендательная система строилась лишь на предложении наиболее популярных видео в приложении, то есть была чисто коллаборативной, никаким образом не учитывая индивидуальные предпочтения пользователя. На данный же момент сложная рекомендательная система, использующая разнообразные механизмы, использует в связке оба подхода к рекомендациям, выстраивая рекомендации как за счёт оценок пользователей с похожей историей взаимодействий с видеоконтентом, так и за счёт учёта разнообразных свойств видео. Очевидно, что сложную рекомендательную систему «Ютуба» невозможно написать в одиночку или маленькой командой, однако их подходы к тщательной регуляции рекомендаций и использование самых современных методов машинного обучения стоит учитывать[7].

1.2.3 Стриминговый сервис Netflix

Не менее интересны подходы стриминговых сервисов, на данный момент они обладают высокой популярностью среди пользователей и очень активно конкурируют между собой. В этой конкуренции немаловажную роль играют не только предлагаемые услуги и каталог контента, но и умелая рекомендательная система, которая с большей вероятностью предлагает то, что понравится пользователю. Именно с умелой системой рекомендаций связывают успех Netflix ещё до коронавирусной пандемии.

Показательным примером того, насколько большую роль для Netflix

всегда играла рекомендательная система, является конкурс *The Netflix Prize*, проведённый в 2008 году. Данный конкурс дал сильный толчок к развитию рекомендательных систем, его целью было разработать наиболее эффективную рекомендательную систему на рынке, способную работать с огромными базами данных, которая обгонит существующие рекомендательные системы, уменьшив среднеквадратичную ошибку обучения моделей на 10 процентов. То, что на словах кажется простой задачей, в итоге растянулось на целых три года, все три года осуществлялся поиск решения.

В итоге на третий год команда разработчиков *Bellkor's Pragmatic Chaos* предоставила алгоритм рекомендательной системы, составленный из 207 различных вычислений, в том числе алгоритмов машинного обучения и поиска по датасету, который справлялся с поставленной целью. В итоге Netflix выплатила обещанный миллион призовых разработчикам, однако так и не внедрили данную модель в связи с очень большой сложностью её реализации. В итоге были внедрены лишь некоторые алгоритмы. С тех пор прошло много времени и были внедрены многие другие алгоритмы обучения, от уже устаревших «трансформеров» до самых современных моделей, основанных на тех же нейросетях [5].

Важной частью в терминологии рекомендательной системы от Netflix является система поощрений или подкреплений для моделей машинного обучения. Данная система позволяет понять моделям, насколько её рекомендация оказалась релевантной (актуальной) пользователю: система поощрений запоминает набор свойств и записывает их в историю взаимодействий. Эти свойства включают в себя как простые, такие как нажатие на рекомендацию (значение 0, если не нажал и 1, если нажал), так и более сложные способы отслеживания реакции пользователя.

Одним из показательных алгоритмов, разработанных за долгие годы функционирования данного стримингового сервиса, является система отложенного фидбека, схематично изображенная на рисунке 1.2, которая собирает долгосрочную информацию о реакции пользователя на содержание, в том числе информацию о том, как быстро он просмотрел весь сериал, был ли им просмотрен сезон до конца, его динамику оценок от серии к серии. Вся информация тщательно обрабатывается большим набором сложных алгоритмов машинного обучения в качестве объекта поощрения. Очевидно, что в малом приложении реализация такого рода системы маловероятна [8].

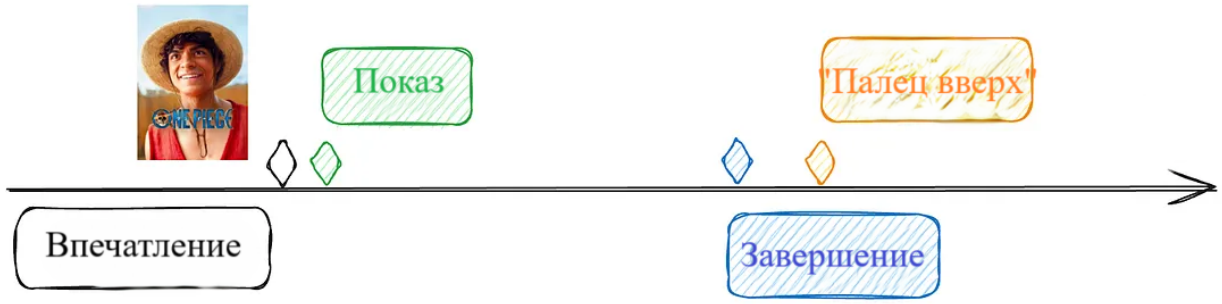


Рисунок 1.2 – Схема отложенного фидбека — реакция пользователя на рекомендацию отслеживается в долговременной перспективе, начиная от начала показа видео и заканчивая окончанием просмотра и полученной реакцией пользователя.

1.3 Итоги сравнения коммерческих рекомендательных систем

По итогам сравнения рекомендательных систем трёх рассмотренных выше приложений будет просто структурировать рассмотренные выше данные в виде таблицы 1.1, которая представлена ниже.

Критерий	**Ozon**	**YouTube**	**Netflix**
Тип системы	Гибридная с преобладанием контент-фильтрации	Гибридная (коллаборативная + контент-фильтрация)	Гибридная (сложные ML-модели)
Основные данные	– История просмотров и покупок – Демография пользователя – Характеристики товара – Покупка рекламы	– История просмотров – Время просмотра – Лайки/дизлайки – Опросы – Данные Google	– История просмотров – Скорость просмотра – Оценки эпизодов – Долгосрочные паттерны
Алгоритмы	– Градиентный спуск – Ранжирование по релевантности	– Глубокое обучение – Анализ поведения – Система поощрений	– Ансамбли из 200+ алгоритмов – Система поощрений – Отложенный фидбек
Особенности	– Учет интересов продавцов – Финальное ранжирование с учетом бизнес-логики	– Борьба с кликбейтом – Мультиплатформенность	– Долгосрочный анализ – Конкурс Netflix Prize
Эволюция	От контентной к гибридной	От коллаборативной к гибридной	Постоянное усложнение моделей

Таблица 1.1 – Сравнительный анализ рекомендательных систем Ozon, YouTube и Netflix

1.4 Анализ алгоритмов фильтрации на основе содержания

Прежде чем приступить к дальнейшему проектированию архитектуры приложения, необходимо разобрать конкретные подходы к формированию рекомендаций на основе содержимого с использованием упомянутых выше математических алгоритмов. В данном анализе, в отличие от анализа выше, будет учитываться также специфика рекомендательной системы разрабатываемого приложения, а именно его предметная область и планируемая цель.

Задача состоит в том, чтобы рекомендовать философские произведения, близкие к тем, которые пользователь оценил положительно. При этом важно учитывать семантическую близость текстов (аннотации, цитаты, темы) и темпорально-тематическую близость школ и эпох. В частности, если пользователь увлекается трудами платоников, можно предположить интерес и к другим представителям данной школы или близким по духу авторам. Для решения этой задачи в основном используются современные модели представления текста: контекстуальные языковые модели (например, BERT), а также статические эмбеддинги (библиотеки FastText, Doc2Vec и так далее), которые векторизуют содержание текстов и позволяют вычислять расстояние между ними. Кроме того, необходимо уметь обрабатывать категориальные признаки (школу, эпоху, страну) и интегрировать пользовательские рейтинги в модель[9].

1.4.1 Текстовые эмбеддинги

Для анализа текстового контента философских работ и связанных описаний используются методы векторизации текстов. Ранее часто применялись «мешок слов» и TF-IDF, однако они не учитывают лингвистическую семантику и дают разреженные представления. Так, классическая модель TF-IDF описывает текст простыми счетными признаками слов, «независимыми» друг от друга. Это удобно и интерпретируемо, но, как отмечают исследования, она не отражает семантическую связь между словами[10]. Функция для расчёта TF-IDF:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$$

где $\text{tf}(t, d)$ — частота термина (term frequency) в документе, $\text{idf}(t)$ — обратная частота документа, то есть частота встречаемости этого слова среди документов.

В частности, при помощи TF-IDF-сравнения невозможно обнаружить, что «естествознание» и «биология» близки по смыслу, в случае если они не совпадают буквально. Для преодоления этого ограничения применяются

плотные эмбеддинги слов и документов. Эмбеддинг — это числовой вектор фиксированной длины, который представляет семантическое значение входного текста.

Одним из наиболее распространённых подходов являются алгоритмы Word2Vec и FastText, обучаемые на большом корпусе текстов. Они выдают для каждого слова (или n-граммы) низкоразмерный вектор, так что семантически близкие слова располагаются рядом в пространстве. Первый также разбивает слова на подслова, что позволяет получать векторы для редких или сложных слов, и это позволяет решить проблему out-of-vocabulary (проблема, при которой сложные и редкие слова разбиваются на отдельные символы, что не позволяет получить их корректный семантический анализ). Полученные векторы слов можно агрегировать (например, усреднением с весами TF-IDF) для получения вектора всего документа или описания. К примеру, набор слов с эмбеддингами [1,2,3], [4, 5, 6] и [7, 8, 9] при усреднении $\frac{1+4+7}{3} + \frac{2+5+8}{3} + \frac{3+6+9}{3}$ будет представлять единый вектор [4,5,6]. Такие представления отражают смысловую составляющую текста: например, расстояние между векторными представлениями «счастье» и «радость» будет невелико [10].

Более совершенные результаты дает использование трансформеров, таких как BERT (Bidirectional Encoder Representations from Transformers). Это языковая модель, предобученная на больших корпусах, которая генерирует контекстуальные векторы слов и предложений. BERT учитывает порядок слов и контекст обеих сторон (левого и правого) при формировании эмбеддингов, что особенно важно для многозначных философских терминов. Полученные эмбеддинги можно усреднять или использовать специальные токены для представления всего текста. Преимущество BERT в том, что он глубоко улавливает семантику и синтаксическую структуру, позволяя лучше сопоставлять тексты с близкими смыслами (например, разные формулировки одной философской идеи). Недостаток – высокая вычислительная сложность и необходимость адаптации под предметную область (понимание терминологии). В большинстве случаев модели на базе трансформеров дообучают на корпусе философских текстов для повышения качества.

Наконец, метод Doc2Vec позволяет сразу строить векторы целого документа. Он по сути расширяет идею Word2Vec на документы: модель обучается так, чтобы предсказывать слова документа с учетом общего «вектора документа». В результате каждому философскому произведению сопоставляется фиксированный вектор, отражающий тематику и стиль. Doc2Vec проще настроить, но он менее эффективен по сравнению с новейшими моделями на больших данных: он не учитывает контекст так глубоко, как трансформеры, и чувствителен к выбору параметров обучения. Все описанные подходы сводят текстовые описания к векторам фиксированной

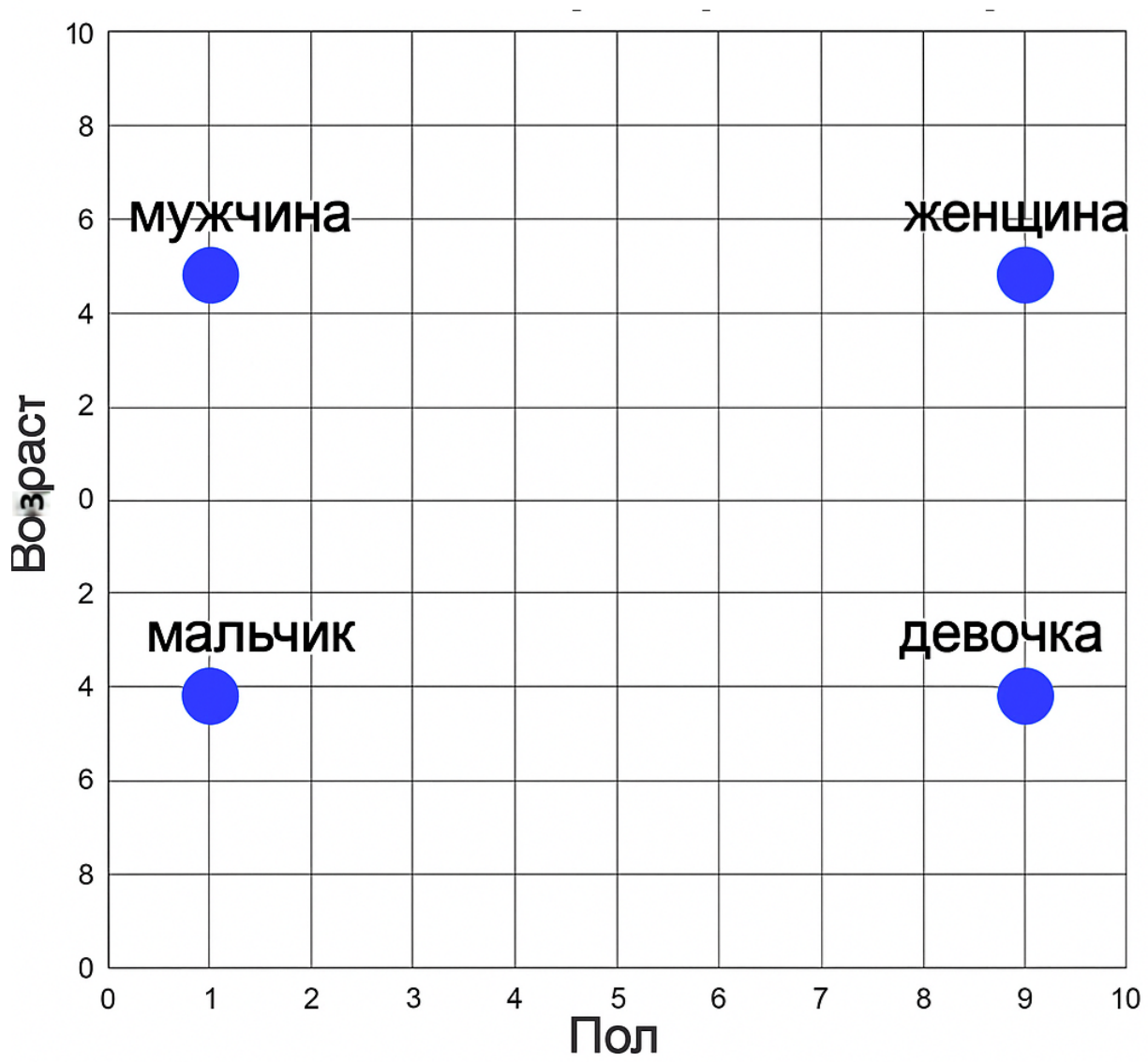


Рисунок 1.3 – Пример векторного пространства семантических признаков для набора слов. Слова со сходным смыслом группируются рядом на основе их значений по скрытым признакам (например, «возраст» и «пол»)

размерности, что даёт возможность сравнивать их стандартными метриками расстояния или схожести. На рисунке 1.3 иллюстрируется простейший случай: в двумерном пространстве «семантических признаков» слова с близким смыслом располагаются рядом (так, «мужчина»/«мальчик» объединены осью «пол/возраст», а «женщина»/«девочка» – аналогично). Это упрощённое изображение демонстрирует идею эмбеддинг-пространства, где «семантические оси» задаются скрытыми факторами языковой модели.

1.4.2 Подход к обработке категориальных метаданных

Помимо текста, важно учитывать и структурированные метаданные философов и их произведений. К таким категориальным признакам отно-

сятся: школа философии, исторический период, страна происхождения, год рождения и пр. Простое включение этих признаков в профиль пользователя повышает релевантность рекомендаций («если нравятся античные авторы, приоритет — другим антикам»). На практике категориальные признаки кодируют разными способами. Наиболее очевидный – one-hot-кодирование или вклад в векторную эмбеддинг-репрезентацию. Например, каждой школе философии можно сопоставить отдельную размерность вектора (1, если философ принадлежит этой школе, иначе 0). Аналогичным образом год рождения можно использовать как числовой признак (или разбить на эпохи). Этот подход прост, но при большом числе категорий приводит к разреженным векторам[11].

Другой путь – использование графовых моделей или графа знаний. Составляется граф, где вершины – философы и их работы, а ребра – отношения «написал», «принадлежит к школе», «влияние» и так далее. В таком графе можно применять алгоритмы разметки/эмбеддинга узлов, чтобы получить векторы философов и работ, учитывающие их атрибутивные связи. Например, модель может обучаться так, чтобы философы одной школы имели близкие эмбеддинги. Недостаток графового подхода – необходимость вручную задавать связи и затратная модель, но он эффективен при разреженности данных и сложных взаимосвязях[12].

1.4.3 Метрики сходства

После того как объекты и профиль пользователя представлены наборами признаков (векторные эмбеддинги + категориальные свойства), выбирают метрику схожести, чтобы сравнить их. Наиболее распространена косинусная метрика: она измеряет угол между векторами и нормирована от -1 до 1, высокое значение означает сильное сходство:

$$\mathrm{sim}(u, v) = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2}\sqrt{\sum_{i=1}^{n} v_i^2}}$$

где A_i и B_i — i-е компоненты векторов A и B, соответственно.

Косинусную метрику часто рекомендуют при работе с высокоразмерными векторами признаков, так как она нечувствительна к их длине. Альтернатива – евклидово расстояние или обратная величина расстояния (чем меньше расстояние, тем выше похожесть). Для категориальных унитарных признаков (one-hot) можно применять коэффициент Жаккара или простое пересечение [13]. Стоит отметить, что для полной оценки сходства иногда объединяют несколько метрик. Например, текстовую семантику (из

BERT) можно комбинировать с категориальной близостью.

1.5 Профиль пользователя

Пользовательский профиль в контентных системах формируется на основе исторических оценок пользователя (лайков/дизлайков просмотренных произведений). Простейший способ – усреднить векторы эмбеддингов произведений, которые пользователь отметил лайком, и получить итоговый «профильный» вектор. При этом отрицательные оценки (дизлайки) можно учитывать с обратной полярностью, уменьшая вес соответствующих признаков. В дальнейшем каждый новый пользовательский фидбэк обновляет профиль: например, при попадании в понравившиеся автоматически пересчитывается центр масс. При таком подходе профиль всегда «тянется» в сторону предпочтительных тем[14].

Профиль пользователя обычно содержит векторы тех элементов, которыми он интересовался ранее (лайки, дизлайки, оценки). Это позволяет рекомендовать новые объекты, близкие к уже оценённым. Практически метрики применяются именно к этому профилю: ищут произведения с максимальным косинусом от профиля. При появлении новых данных профиль динамически смещается. Такой онлайн-обучаемый профиль способен адаптироваться к изменению интересов, хотя и рискует «заклинить» на избранной тематике, уменьшая серендипность и создавая пузырь фильтраций [13].

1.6 Вывод

В результате проведенного анализа рекомендательных систем возможно сформулировать несколько выводов. Первый из них — формулирование требований к рекомендательным системам, которые предъявляют к последним потребители и разработчики:

– **Производительность** — важная часть рекомендательной системы. Будучи довольно затратной частью приложения, важно, чтобы вычисления не понижали его скорость и не затормаживали выполнение остальных запросов, поэтому многие компании выносят все расчеты для рекомендательных систем в отдельные сервисы. При этом при заботе о производительности не стоит забывать и про эффективность — среднаквадратичная ошибка, основная метрика точности конечного результата, должна быть минимальной.

– При разработке рекомендательной системы также важно учесть **разнообразие рекомендаций**, так как часто в результате вычислений может получиться так, что в выборку попадают произведения от 1-2 исполнителей, когда пользователь ожидает рекомендацию в том числе и новых позиций. Для этого многие компании ставят ограничения по свойствам на

выборку рекомендаций по умолчанию, стараясь не рекомендовать продукты от одного производителя.

– Проблема **приватности** является важной составляющей успеха у большой аудитории. Рекомендательная система запрашивает и хранит очень много информации о пользователе, многим важно иметь уверенность в том, что их данные не попадут в третьи руки и не будут использованы против них. Поэтому часто компании предварительно запрашивают разрешение на использование данных и ставят разнообразные дополнительные механизмы защиты на базу с данными для предотвращения утечек информации.

– Очень интересной частью удачной рекомендательной системы является **интуативность** с более удачным и точным по смыслу английским аналогом *serendipity* (серендипность), которая обозначает неожиданное, но удачное открытие в случайных обстоятельствах. На примере рекомендательной системы можно привести такой пример: если онлайн-магазин продуктов рекомендует покупателю молоко, то это является правильно рекомендацией, однако вряд ли интересной пользователю в связи с её очевидностью. Удачная система рекомендаций должна предоставлять пользователю возможность делать открытия прямо во время пользования сервисом, открывая для себя нечто приятное и нужно, о чём он раньше никогда не думал. Выше данный термин упомянут в контексте формирования профиля пользователя.

– При построении модели важно также учитывать *демографию пользователя*, в том числе его возрастную группу, национальность и основной язык.

– Из анализа становится заметной также необходимость отслеживать **действия пользователя в долгосрочной перспективе**, запоминая его взаимодействия с продуктами на протяжении времени, например учитывая, если в будущем он убрал с него лайк или повысил ему оценку.

Из технических особенностей реализации рекомендательной системы, основанной на содержимом, стоит отметить необходимость комбинирования подходов: эмбеддинги слов и предложений с помощью BERT и Doc2Vec обеспечивают качественное представление смыслового содержания аннотаций, цитат и биографий. Категориальные признаки выступают дополнительными свойствами, усиливающими тематическую согласованность рекомендаций. Метрики сходства, прежде всего косинусная, служат фундаментом для сопоставления профиля пользователя с описаниями произведений. Каждый из технических подходов, рассмотренных в анализе, имеет свои плюсы и ограничения. Эмбеддинги BERT дают глубокую семантику за счет вычислительных затрат; FastText/Word2Vec – лёгки, но менее точны; Doc2Vec – прост в идее, но устаревший. Категориальные признаки обогащают модель, но требуют аккуратного кодирования (в простейшем варианте – one-hot вектора или расширения пространства признаков). Гибридные подходы, сочетающие разные источники информации (текст + метаданные + граф),

в литературе считаются наиболее эффективными.

На текущем этапе развития рекомендательных систем построить систему, которая удовлетворяет всем вышеописанным требованиям, уже не является настолько большой проблемой. И хотя проанализированные выше модели с их высоконагруженными алгоритмами практически невозможно реализовать маленьким коллективом, многие готовые решения готовы предложить более простой аналог за куда меньшие трудозатраты и с меньшей нагрузкой на приложение. В рамках выбранной темы — философии, аналогов рекомендательных систем фактически нет, что как усложняет задачу, заставляя с нуля протаптывать неизученную тропу, так и упрощает, несколько понижая требования пользователя к будущей системе.

2 ПРОЕКТИРОВАНИЕ МОДЕЛИ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ ПО ФИЛОСОФИИ

Одной из важных вещей является планирование дальнейшей разработки, недостаток которого может привести к большим проблемам по ходу написания кода, тестирования и развертывания программы в открытом доступе.

Так как рекомендательная система будет полноценным веб-приложением, очень важно организовать взаимодействие клиента с базой данных, аккуратно и точно расписать бизнес-логику и внедрить в неё алгоритмы бизнес-систем. Для этого нужно предварительно выбрать архитектуру будущего приложения и разработать на основе этой архитектуры действующую модель.

Перед началом разработки нужно определиться со средствами разработки, изучить их возможности и способы реализации в них нужного шаблона проектирования. При выборе нужно ориентироваться на личные предпочтения, уровень владения инструментами и задачи, которые требуется выполнить по ходу конструирования продукта. Также часто, так происходит и в данном случае, средства разработки выбирает заказчик. В данном случае нам нужно ознакомиться с тем, что будет помогать нам реализовать бизнес-логику, пользовательский интерфейс и где будут храниться данные веб-приложения. При анализе подходов было выяснено, что при реализации рекомендательной системы очень желательно придерживаться микросервисной архитектуры с целью не нагружать вычислениями общий модуль приложения.

2.1 Постановка задачи

Итак, в рамках разработки рекомендательной системы по философии нужно определить шаги по разработке точной и производительной архитектуры не только как веб-приложения с быстрыми запросами, не нагружающими память сервера и соблюдающими основные соглашения и подходы к разделению обязанностей, устойчивости и расширяемости веб-приложений.

Начать стоит с понимания того, какой именно поиск и какая цель рекомендательной системы будут основной, на чем будет основной фокус приложения и какой функционал будет предложен пользователям. Разрабатываемое в будущем приложение должно будет осуществлять поиск по пяти основным сущностям:

– Персоналии (философы и мыслители);

– центральные работы философов, в том числе лекции эссе и книги, умело категоризованные по персоналиям и разделам и направлениям философиию В случае, если работы философов находятся в открытом доступе, их текст будет размещен на сайте с возомжностью прочтения, также текст будет анализироваться рекомендательной системой;

– разделы философии, такие как идеализм, материализм, экзистенциализм и так далее, в том числе их частные подвиды. Они являются еще и ключевой опорой, которой позволит рекомендательной системе разбивать сущности на группы в рамках, к примеру, кластерного анализа. При этом задача категоризации на разделы является самой нетривиально и многогранной из всех, так как даже единого деления на основные подветви часто расплывчато и туманно трактуется даже самими философами, что отлично заметно на примере рисунка 2.1;

– концепции философии — часто одна концепция, к примеру трансцедентальность, диалектика и витализм, проходит через множество философских школ;

– цитаты, которые в условиях текущего быстрого и жестокого к свободному времени мира, являются основным способом вовлечения. Людям куда удобнее прочитать 5-10 цитат, чем читать даже отрывок из комплексного философского трактата.

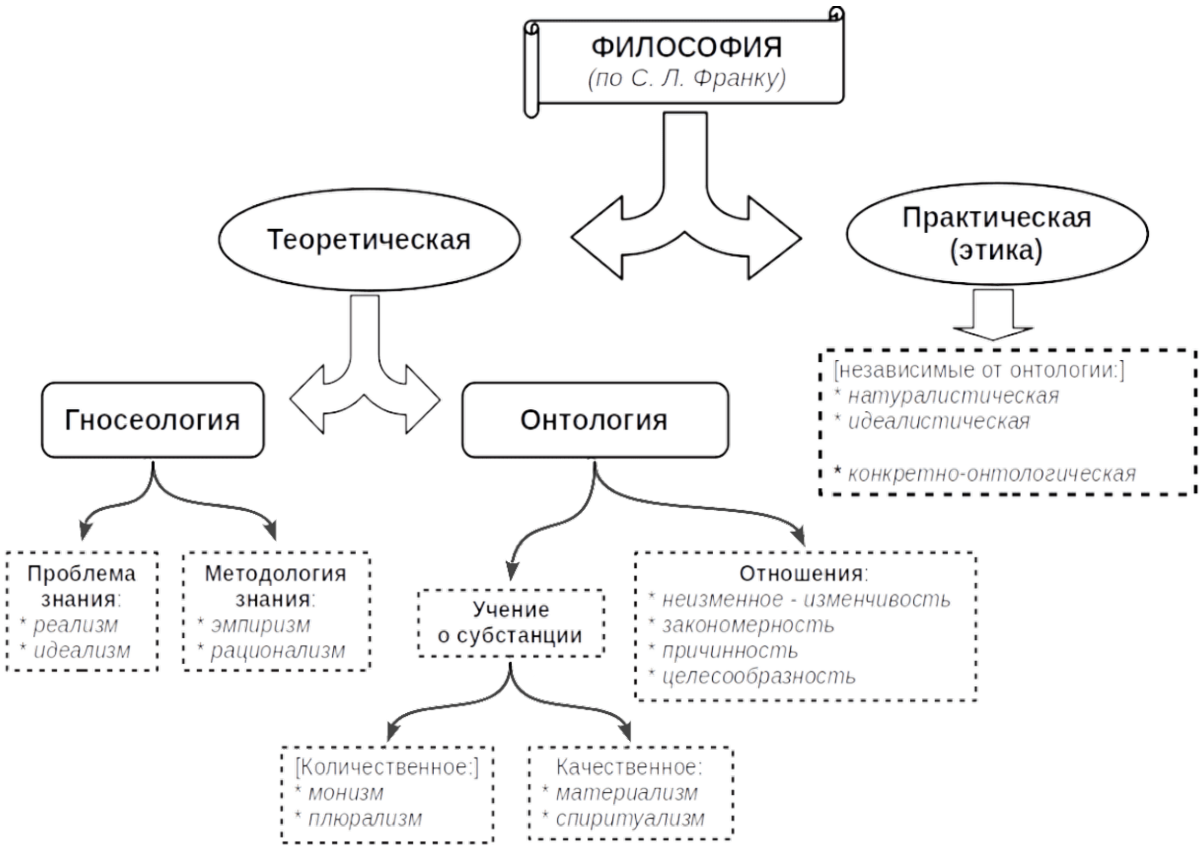


Рисунок 2.1 – Взгляд на базовую категоризацию философии от Семёна Франка, одного из важнейших философов 20 века.

Понимая данный фронт работ, формируется предметная область, вокруг которой строится приложение, на основе этой области строится вся база знаний с продуманными связями, при этом в процессе создания данной предметной области можно отталкиваться и от открытых источников с категоризацией, к примеру от Википедии. Надо понимать, что философские тексты и цитаты требуют тщательной разметки: помимо базовых метаданных вроде автора и исторического периода, необходимо выделять школы мысли (стоицизм, экзистенциализм), ключевые концепты («категорический императив», «диалектика») и отношения между работами (критика, развитие идей). Для этого можно комбинировать структурированные источники вроде академических энциклопедий с автоматизированным анализом полных текстов. Например, обработка цитат из «Критики чистого разума» может выявить связи с более поздними трудами неокантианцев, а семантический анализ поможет соотнести работы Хайдеггера и Сартра через обсуждение экзистенциальной свободы. И через тщательную обработку цитат уже дальше находить способы сделать более продуманную вывод рекомендаций пользователю приложения.

На ранних этапах нет смысла активно реализовывать коллаборативную фильтрацию в связи с её зависимостью от других пользователей, при малом числе пользователей она будет вносить лишь раздражающие

искажения в конечную модель, которая лишь снизит удовлетворённость программой. На начальных этапах приложения и, с учётом специфики и многообразия предметной области, и на поздней части, стоит ориентироваться на *фильтрацию на основе содержимого* с применениями алгоритмов машинного обучения, которые были разобраны в прошлом разделе.

Для эффективной фильтрации на основе содержимого очень важно поверх умелой фильтрации по категориям и свойствам стараться организовать фильтрации на основе паттернов в текстах работ философов. Для обработки текстов можно использовать и традиционные методы вроде TF-IDF, которые могут выделить частые термины, но они не уловят нюансы многозначных понятий, часто встречающихся в философии. Внедрение контекстных моделей, таких как *BERT*, позволяет различать употребление термина «материя» в контексте марксистской диалектики и средневековой схоластики. Схематично процесс анализа текста от BERT изображён на рисунке 2.2. Для работ на разных языках может потребоваться кросслингвистический анализ, поддерживаемый этой моделью: например, сопоставление немецкого понятия «Dasein» (вот-бытие в философии Хайдеггера) с его интерпретациями в англоязычной феноменологии через векторные представления слов. Данная часть системы тяжело реализуема, однако полезна в дополнительном анализе информации и построении конечной модели [15].

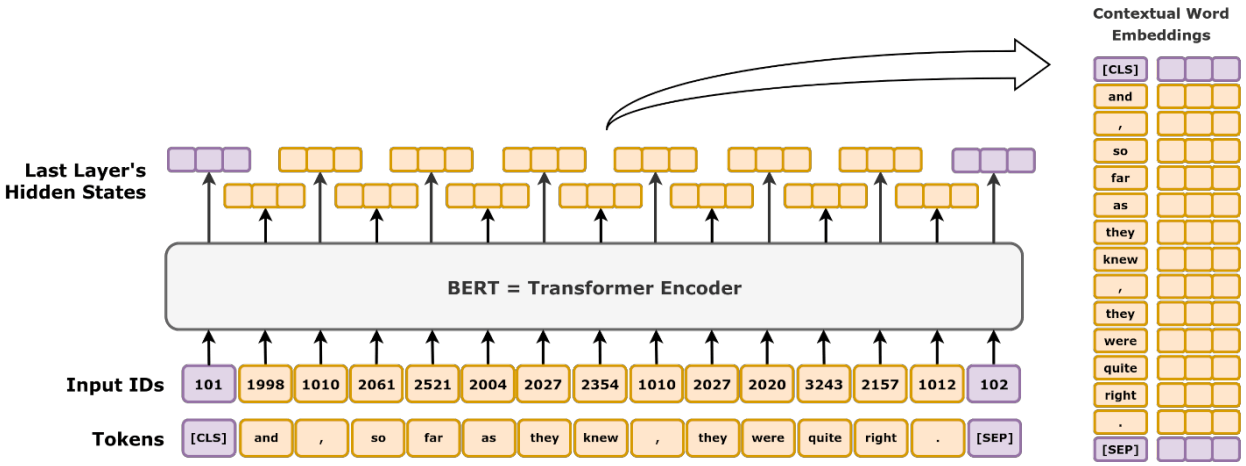


Рисунок 2.2 – Высокоуровневая диаграмма языковой модели BERT. Она берёт текст, токенизирует его в последовательность токенов и кодирует его, чтобы потом внедрить в модель как эмбеддинг текста (его репрезентацию в векторном виде).

При фильтрации на основе содержимого стоит ориентироваться как на категоризацию по году, философу и направлению философии — и так очевидно, что данные характеристики стоит учитывать в зависимости от количества взаимодействий с ними клиента, но учитывать рассмотренные в разделе 1 метрики интеракции, такие как продолжительность просмотра рекомендованной страницы философа, работ, цитат, лайки клиента на них,

это позволит создать качественный механизм подкрепления для модели, без которого она вряд ли будет достаточно точной.

Важно учитывать также эффект «серендипности» (интуативности, упомянутой в выводе в разделе 1.6), который является важным показателем качества рекомендательных систем: важно отслеживать, не создаёт ли система «фильтрующий пузырь», постоянно предлагая философов только одной школы, и встроить механизмы, целенаправленно вводящие элементы разнообразия. Например, после рекомендации нескольких текстов аналитической философии система может предложить контрастный материал из континентальной традиции или же даже из восточных направлений. В таком случае пользователь будет постоянно вовлечён в приложение, не уставая от однотипных рекомендаций.

2.2 Выбор архитектуры приложения и её проектирование

Это критически важный аспект разработки ПО, поскольку от него зависят качество, гибкость и долговечность системы.

Во-первых, архитектура ПО определяет общую структуру системы, что позволяет лучше понимать и управлять её сложностью. Она обеспечивает четкое разделение на модули, упрощая их разработку, тестирование и сопровождение. Хорошо спроектированная архитектура позволяет разработчикам легко добавлять новые функции и изменять существующие, не нарушая работу системы в целом.

Во-вторых, архитектура ПО играет ключевую роль в обеспечении производительности и масштабируемости системы. Она помогает определить, как различные компоненты взаимодействуют друг с другом и как распределяются ресурсы, что влияет на скорость работы и способность системы обрабатывать увеличивающиеся объемы данных и нагрузки.

Безопасность также является важным аспектом архитектуры ПО. Она включает механизмы защиты данных и предотвращения несанкционированного доступа, что критически важно в современных условиях киберугроз.

Кроме того, архитектура ПО способствует упрощению коммуникации между разработчиками, аналитиками и другими заинтересованными сторонами. Она предоставляет общую основу для обсуждения требований, планирования и управления проектом, что улучшает координацию и снижает риски ошибок и недоразумений.

Для упрощения проектирования проекта были разработаны стандартные архитектурные шаблоны проектирования приложений, среди которых как основные выделяются **Model-View-Controller**, **Model-View-Presenter** и **Model-View-ViewModel**, сравнительные диаграммы которых представлены на рисунке 2.3 [16].

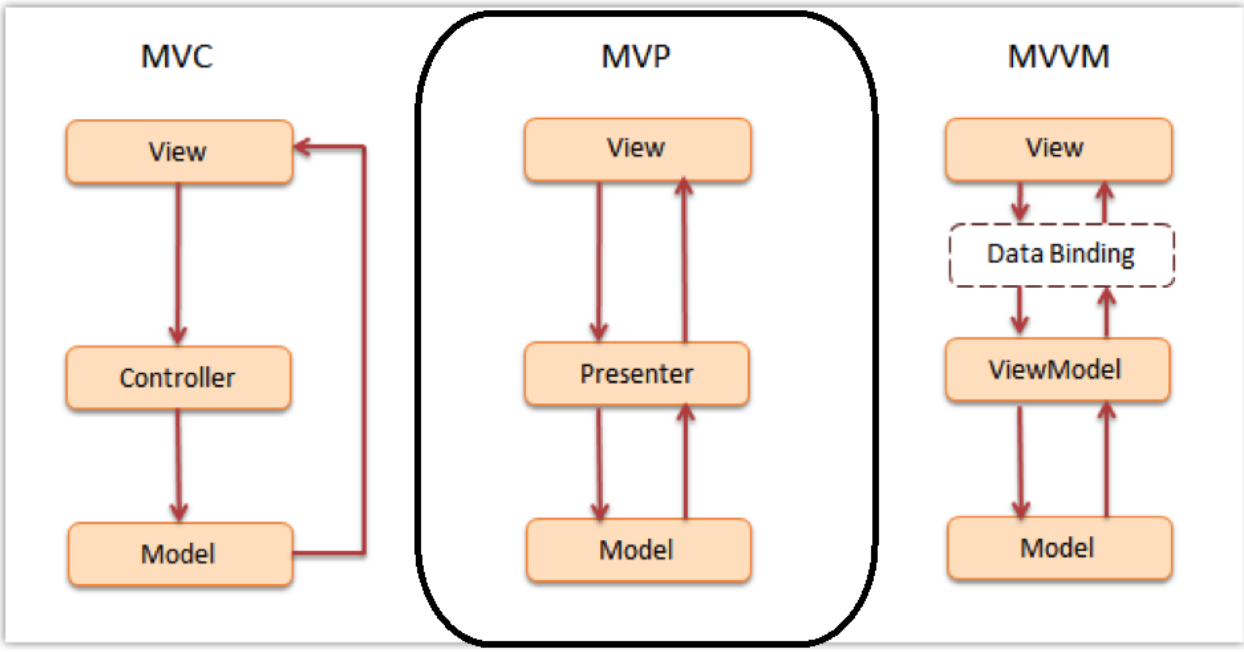


Рисунок 2.3 – Сравнительная диаграмма трёх основных архитектурных шаблонов проектирования. Выбранный в будущем архитектурный шаблон выделен на графике.

После того, как разработчик начинает понимать, каким образом отличаются известные архитектурные паттерны, он должен определить требования, предъявляемые к проекту, это в том числе включает такие шаги, как[16][16]:

1. **Определение типа приложения.** Веб-приложение, мобильное приложение, десктопное приложение и т.д. К примеру, в десктопных приложениях много популярных решений на основе MVVM (WPF, Xamarin), чего не скажешь о веб-приложениях, где MVVM только начинает набирать популярность.

2. **Определить целевую аудиторию.** Определение целевой аудитории (студенты, школьники, пенсионеры и так далее), которая будет пользоваться приложением, — крайне важная часть разработки. Таким образом будет понятно уровень овладения аудиторией технологиями, из которой исходит то, насколько простой требуется интерфейс; будут определена платформа, которую эта аудитория и предпочитает и множество чего другого. В итоге будет сформирована целевая задача продукта, которая в дальнейшем поможет и при выборе оптимальных средств разработки.

3. **Выяснить целевую платформу.** Определить для какой разрабатывается приложения: Android, iOS, Windows или она вовсе будет кросс-платформенным веб-приложением. К примеру, в мобильной разработке на Android и iOS MVC не так популярен из-за того, что сложность контроллеров может становиться чрезмерной, поэтому предпочтительней использовать более строгие к содержимому MVP и MVVM. Крайне важно также учесть общие технические особенности платформы, в том числе,

чтобы системные требования продукта были реалистичными для аудитории, под которую он разрабатывается. Плохая оптимизация продукта может сыграть злую шутку — им не сможет воспользоваться большая часть целевой аудитории.

4. **Определить функциональные требования и сложность проекта.** Маленький проект с простым пользовательским интерфейсом или крупный проект с множеством функций и сложным интерфейсом. Так, например, в простых приложениях оптимальней использовать простой в реализации MVC, однако в сложных приложениях контроллер быстро перегружается и становится неподвластным интеграционному тестированию и здесь уже лучше отдавать предпочтение MVP или MVVM.

5. **Оценить масштабы и сложность проекта.** Требования к производительности, безопасности, тестирумости и поддержке масштабирования. В том числе нужно определить, что для разрабатываемого проекта является наиболее ключевым и играет важнейшую роль, какой подход более оптимален — единый монолит или коллекцией микросервисов, в которой каждый из сервисов отвечает за свой определённый функционал.

Так как текущий проект обладает сложной и комплексной моделью, требующей взаимодействия со сторонним микросервисом со встроенной в него рекомендательной системой, а основная часть приложения будет комплексным веб-приложением со сложным взаимодействием контроллеров, то для разрабатываемого проекта рекомендательной системы по философии стоит остановиться на модели, которая предоставляет более широкое разделение обязанностей и больше подвластна интеграционному тестированию. Лучшим кандидатом для этого является MVP-архитектура, которая не ограничена определёнными средствами разработки, как MVVM, а также не подвержена чрезмерной перегрузке компонентов обязанностями, как вышеупомянутая MVC-архитектура.

2.3 Микросервисная архитектура

В разделах выше была также затронута необходимость обратить внимание на выбор общей архитектуры приложения, выбор необходимо делать между единой монолитной архитектурой и микросервисами. В данном случае необходимо проанализировать плюсы и минусы каждой архитектуры и применить их к требованиям приложения, для понимания какая конкретно подойдёт в текущем случае.

Монолит предлагает следующие преимущества:

– **Простота** — была и остаётся главным преимуществом монолита перед микросервисом. Все компоненты приложения (логика, интерфейс, база данных) объединены в единую кодовую базу, что исключает необходимость настройки сложных механизмов межсервисного взаимодействия.

Это исключает необходимость сложной настройки CI/CD или создания множественных контейнеров с единым форматом сообщений;

– **производительность** — как ни парадоксально, еще один аргумент в пользу монолита. Данный аргумент в основном действителен на маленьких стандартных MVC-приложениях, поскольку все компоненты работают в рамках одного процесса, вызовы между ними происходят напрямую, без сетевых задержек и накладных расходов на сериализацию данных. Однако в условиях крупных параллельных расчётов рекомендательной системы производительность монолита находится под вопросом в рамках тех же ограничений ресурсов контейнера;

– **согласованность** данных в монолите достигается проще благодаря использованию единой базы данных. ACID-транзакции (атомарность, согласованность, изоляция, долговечность) работают «из коробки», что минимизирует риски рассогласования информации. В микросервисной среде, где каждый сервис управляет своей базой данных, обеспечение консистентности требует реализации сложных паттернов вроде Saga или компенсирующих транзакций, что увеличивает сложность кода и вероятность ошибок.

Данные преимущества достаточно заметны и могут повлиять на решения многих разработчиков, однако в условиях множества параллельных расчетов, необходимости использовать несколько языков программирования и отказоустойчивости, а также изолированности рекомендательной системы, перечисленные ниже преимущества микросервисов играют очень важную роль:

– Ключевое преимущество — **децентрализация**. Каждый сервис автономен: он управляет своей бизнес-логикой, базой данных и инфраструктурой, что позволяет командам работать независимо. Например, в крупных приложениях с рекомендательными системами, таких как Netflix или Amazon, разные отделы могут разрабатывать и деплоить функции платежей, рекомендаций или аналитики параллельно, не блокируя друг друга. Так же исключается блокировка расчетов из-за, к примеру, использования одного подключения к базе данных;

– **технологическая гетерогенность** — возможность использовать разные языки программирования, фреймворки и базы данных для каждого сервиса — открывает пространство для инноваций. Команда машинного обучения может внедрить Python и TensorFlow в сервис рекомендаций, в то время как само приложения строится на удобном сочетании средств разработки для веб-приложении C# + ASP.NET;

– **архитектура, удобная для эволюции** — благодаря ей возможность постепенно модернизировать систему становится реальностью. Вместо «большого взрыва» при переходе с монолита можно поэтапно выделять сервисы, начиная с самых проблемных или часто изменяемых модулей. Это снижает технический долг и позволяет адаптироваться к меняющимся

требованиям к рекомендательным системам и веб-приложениям.

В таких условиях становится более обоснованным выбор микросервисной архитектуры с вынесенным в отдельный сервис приложением для расчета рекомендаций, её примерная схема изображена на диаграмме на рисунке 2.4. Эта архитектура позволит разграничить сложные механизмы построения рекомендаций и само приложение, также частично реализуя паттерн *CQRS* (разделения логик получения и обновления данных).

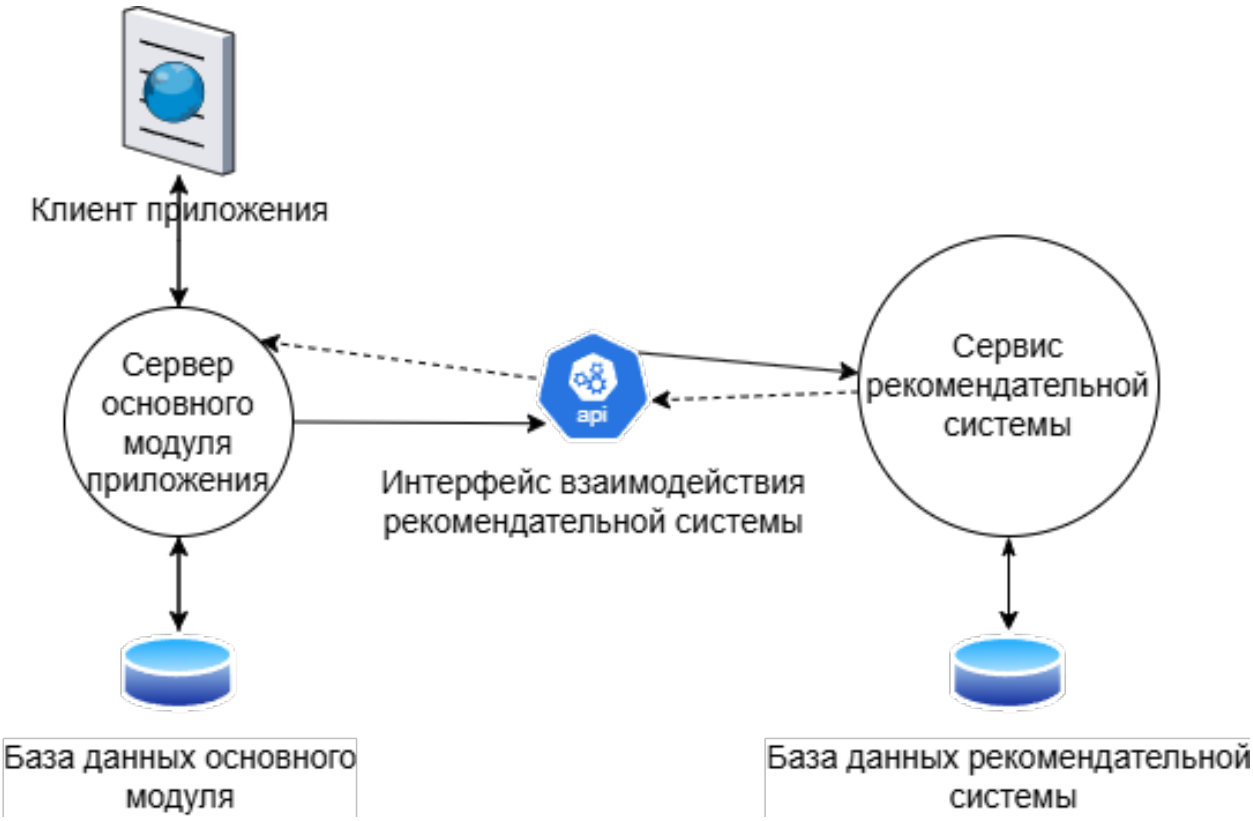


Рисунок 2.4 – Высокоуровневая диаграмма взаимодействия между сервисами приложения.

2.4 Модель архитектуры системы

На основе проведённого теоретического проектирования можно выстроить несколько диаграмм системы, начиная с диаграммы возможных вариантов взаимодействия, представленной диаграммой последовательности на рисунке 2.5.

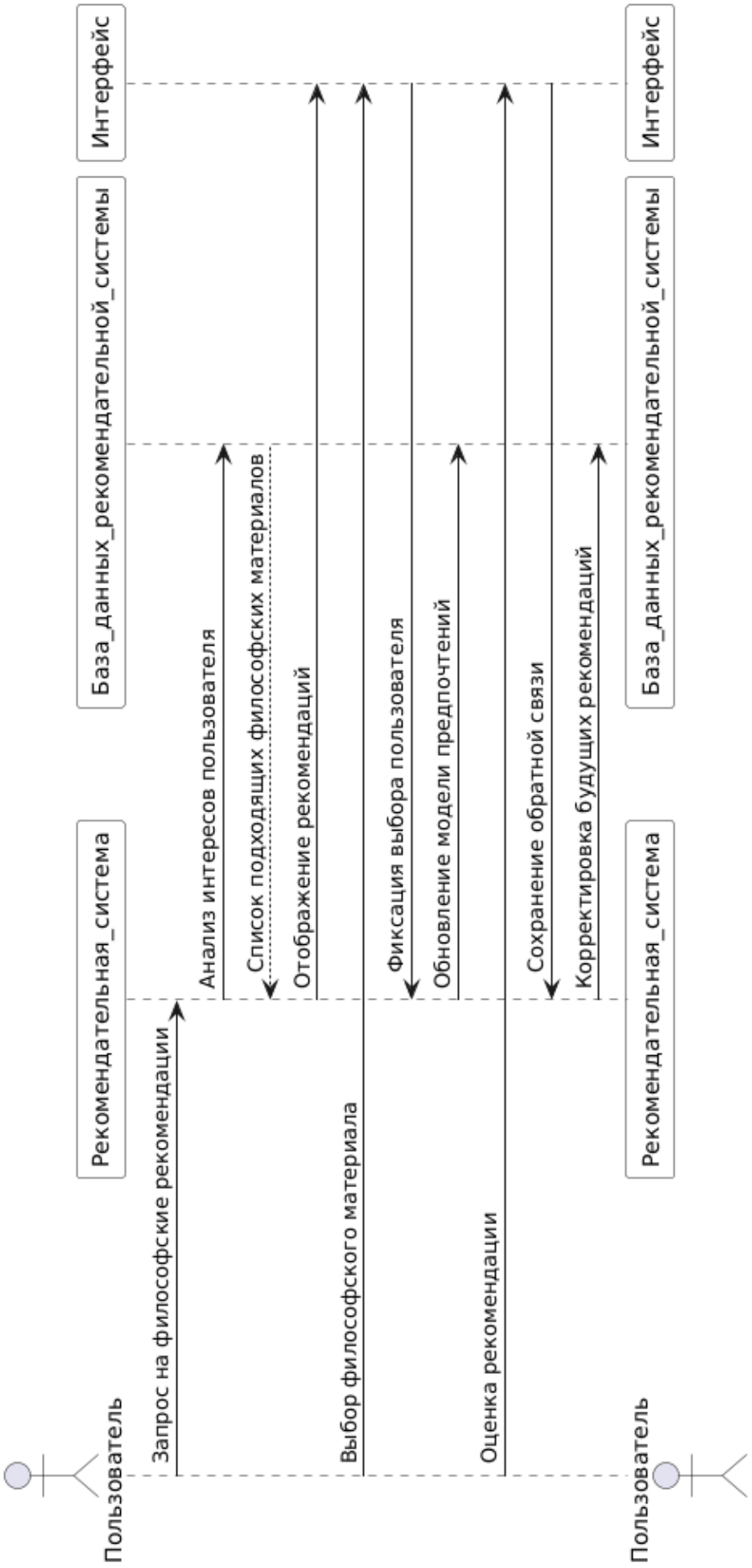


Рисунок 2.5 – Диаграмма взаимодействий приложения.

Диаграмма взаимодействия помогает лучше понять и отобразить требования системы и процесс взаимодействия. Фактически, процесс формирования и отображения рекомендаций будет единственным сложным процессом системы, однако его сложность не является единственной, с чем нам предстоит столкнуться во время разработки. Ещё одним важным этапом, о чём уже говорилось ранее, является формирование правильной схемы представления моделей, для чего необходимым считается построение диаграммы классов, представленной на схеме 2.6:

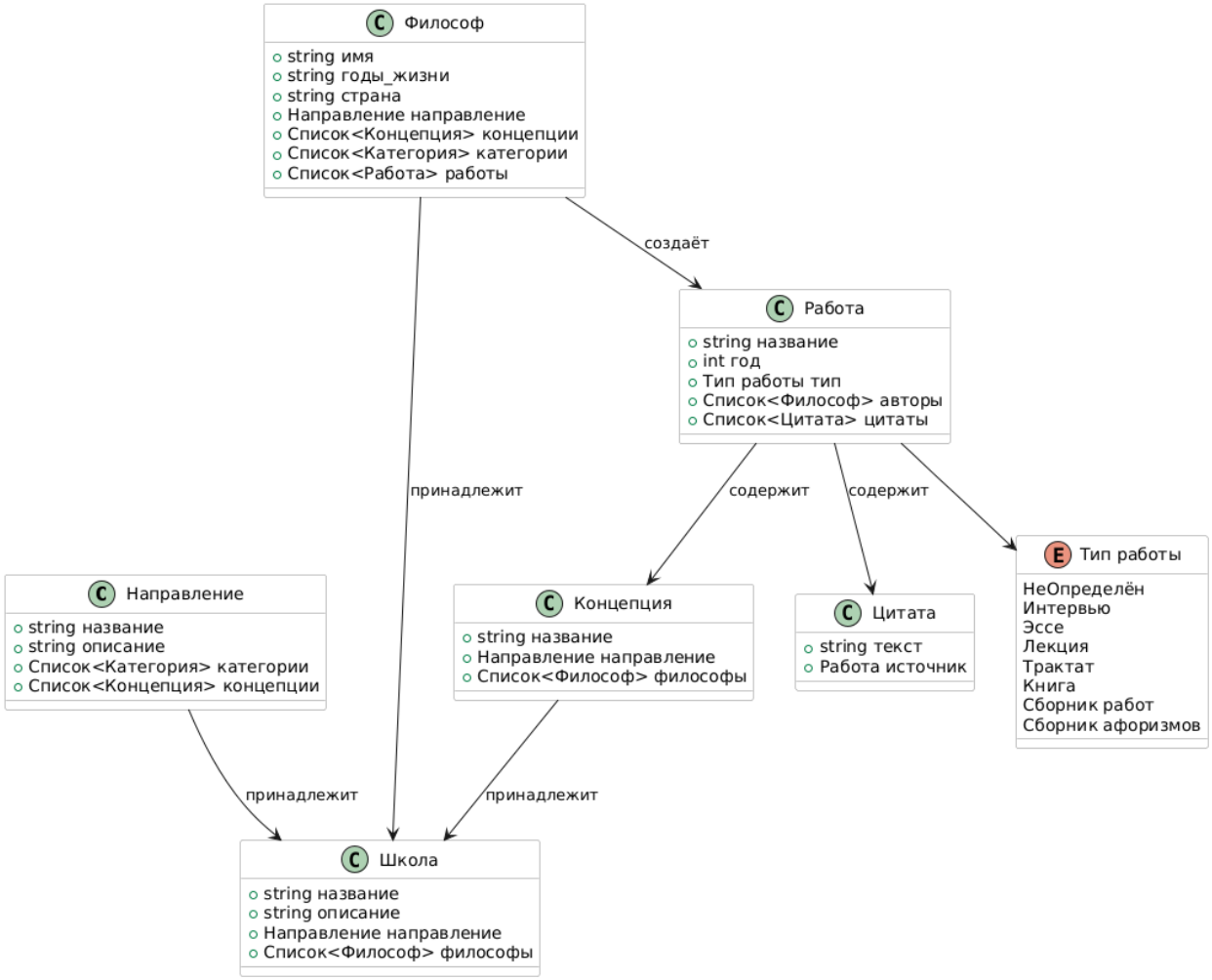


Рисунок 2.6 – Диаграмма классов приложения. Модель сущностей предметной области «Философия». Ассоциативные сущности и рабочие свойства (например, значения векторов, пользователи и их взаимодействия) не отображены.

Для упрощения понимания диаграммы классов в ней опущены некоторые технические признаки и свойства сущностей, в том числе хранимые эмбеддинги и значения векторов, а также профиль и роли пользователя.

Для полного понимания того, как на внутреннем уровне будет устроено приложение и реализована сложная архитектура обмена информацией между компонентами, требуется диаграмма компонентов, которая покажет,

как именно будут реализовываться сформулированные выше требования, подобная диаграмма предоставлена на рисунке 2.7:

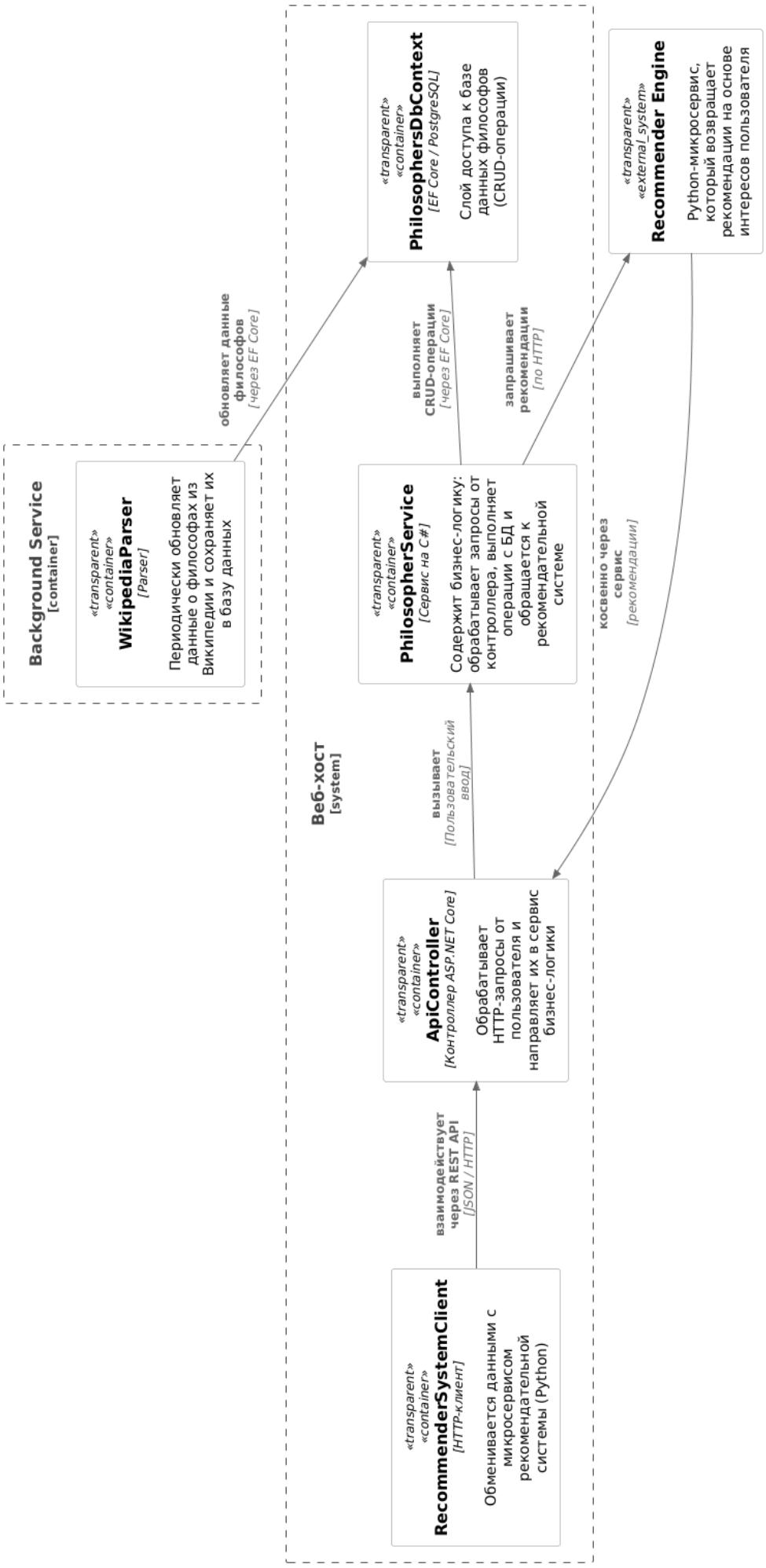


Рисунок 2.7 – Диаграмма компонентов системы

Из этой диаграммы становятся понятны шаги по реализации и разработке будущего приложения. На этом этапе от проектирования можно переходить к разработке и реализации всех вышеупомянутых схем на практике.

2.5 Вывод

По итогам проектирования были сформулированы основные требования к приложению с учётом конечных целей и особенностей предметной области, а также исходя из сформулированных задач. В результате подробного и всестороннего анализа архитектур была продумана архитектура продукта и взаимодействие между сервисами в нём, в дальнейшем осталось лишь определиться с технологиями разработки и используемыми инструментами для этой же разработки.

В результате осуществленного проектирования становятся понятны цели, которые необходимо достигнуть, и пути их достижения, также становятся понятны необходимые для успешной разработки шаги, которые будут выполняться по ходу реализации приложения. Также был проведен полноценный анализ самого главного компонента будущего сервиса — рекомендательная система, определены требования к ней и главные вызовы, стоящие перед разработчиком; на эти вызовы в том числе шла опора при выборе архитектуры в ходе проектирования, и они ложатся крупной нагрузкой на будущее приложение.

Однако по итогу проектирования можно сказать о релевантности и важности приложения, его актуальности в условиях бурно развивающихся технологий, в том числе за счёт интересной и сложной предметной области, дающей разработчику вызов своей комплексностью категоризации в сравнении с любыми другими системами и скрещенной с самыми современными технологиями разработки.

3 РАЗРАБОТКА РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ ПО ФИЛОСОФИИ

3.1 Используемые средства разработки

Во время разработки приложения были использованы разнообразные языки и инструменты. Во многом такое разнообразие стало возможным благодаря микросервисной архитектуре, позволившей использовать независимые экосистемы для предоставления пользователям уникального опыта. Используемые инструменты включают в себя:

1 *ASP.NET Core* — современный фреймворк для разработки веб-приложений от компании Microsoft. Используется для построения основной серверной части системы, а именно API-контроллеров, обеспечивающих взаимодействие пользователя с системой, а также получения и отправки данных.

2 *PostgreSQL* — объектно-реляционная система управления базами данных с открытым исходным кодом. Отличается высокой надёжностью, масштабируемостью и поддержкой сложных запросов, что делает её оптимальной для хранения структурированных данных о философах, работах, цитатах, направлениях и пользовательской активности (например, лайках и дизлайках). Во многом полезна так же и её возможность хранить данные в документоориентированном виде благодаря наличию формата данных *jsonb*

3 *Python* — язык программирования общего назначения, широко используемый в области анализа данных и машинного обучения. Применяется для реализации рекомендательных алгоритмов, обработки пользовательских предпочтений и генерации персонализированных списков рекомендаций.

4 *scikit-learn* — библиотека Python для машинного обучения. Используется для реализации алгоритмов фильтрации на основе содержимого, включая вычисление сходства между объектами и обучение моделей предпочтений на основе истории действий пользователя. Использует в том числе модели, основанные на BERT, для кластеризации предпочтений.

5 *pandas* — библиотека для обработки и анализа данных в табличной форме. Применяется для предварительной обработки информации о контенте и действиях пользователей перед подачей в рекомендательные модели.

6 *NumPy* — фундаментальная библиотека Python для работы с многомерными массивами и математическими операциями. Используется в комбинации с *pandas* и *scikit-learn* для ускорения вычислений и работы с векторами признаков.

7 *FastAPI* — высокопроизводительный веб-фреймворк на Python для создания REST API. Используется в отдельном сервисе для предоставления

доступа к рекомендательным функциям, которые будут вызываться из основного приложения на ASP.NET.

8 *Docker* — платформа контейнеризации, позволяющая запускать все компоненты системы (бэкенд, рекомендательный сервис, база данных) в изолированных средах. Использование Docker обеспечивает предсказуемость поведения приложений, упрощает развертывание и масштабирование проекта.

9 *Docker Compose* — инструмент для одновременного запуска и управления несколькими контейнерами. Используется для организации архитектуры микросервисов: каждый сервис системы (ASP.NET API, Python-алгоритмы, база данных) запускается в отдельном контейнере, что способствует гибкости и модульности архитектуры.

10 *TypeScript* — язык программирования для веб-разработки, в основе которого лежит язык программирования JavaScript. TypeScript позволяет создавать интерактивные веб-страницы, что повышает удобство взаимодействия пользователя с веб-сайтом. Использование TypeScript при реализации курсового проекта также обусловлено тем, что данный язык программирования позволяет избегать ошибок типов, которые могут возникать при разработке на JavaScript, что в свою очередь увеличивает скорость разработки пользовательского веб-интерфейса.

11 *React* — это библиотека JavaScript с открытым исходным кодом для создания пользовательских интерфейсов. Она позволяет разрабатывать удобные и динамичные веб-приложения, предоставляя простой способ работы с компонентами и состоянием.

В данном случае ASP.NET Core будет использоваться для разработки бизнес-логики, то есть базовой серверной части приложения; Python с необходимым набором библиотек будет использоваться для обучения модели, на базе которой будет строиться рекомендательная система; React + TypeScript будут использоваться для разработки стильной и современной клиентской части.

3.2 Выбор модели обработки текстов BERT

Для обработки текстов в системе была выбрана модель BERT, разработанная на основе архитектуры трансформеров. Эта модель обеспечивает высокую точность при анализе текста за счёт двунаправленного подхода к обработке последовательностей, что позволяет учитывать контекст слов как слева, так и справа.

Основные преимущества использования BERT включают способность извлекать глубокие семантические связи из текста, что делает её идеальной для задач поиска, классификации и кластеризации. Благодаря предобучению на большом объёме текстовых данных, модель может эффективно

работать с текстами на естественном языке, включая сложные языковые конструкции и специфические домены.

Выбор BERT обусловлен её универсальностью и производительностью. Она способна генерировать эмбеддинги текста, которые сохраняют его смысл и структуру, что важно для реализации семантического поиска и анализа данных. Наличие предобученных версий модели для различных языков, включая русский, также снижает затраты на её интеграцию в систему.

Кроме того, модель легко адаптируется к специфике данных путём дополнительного обучения на доменных текстах. Это позволяет достичь высокой точности и релевантности в задачах обработки текстов, делая BERT оптимальным выбором для реализации функционала системы.

В приложении модель будет работать вместе с one-hot алгоритмом и косинусными сравнениями[17].

3.3 Реализация компонентов системы

Во всех случаях, очевидно, очередность разработки и подхода к ней играет важную роль. На основе спроектированных в предыдущем разделе примеров архитектуры приложения шаги по дальнейшей проработке взаимодействия между компонентами приложения становятся более ясными.

3.3.1 Разработка модели баз данных

Разработка начинается с создания и заполнения базы данных нашей моделью с помощью использования ORM-систем. На .NET самой популярной является Entity Framework, с помощью которой можно легко сформировать схемы и таблицы для нужных моделей в реляционной базе данных и обеспечить взаимодействие с ней, не составляя при этом сложных запросов на языке SQL.

Один из важнейших вызовов, стоящих на данном этапе — правильный способ хранения информации о действиях пользователя для обеспечения правильных алгоритмов фильтрации контента для модели рекомендательной системы. В основном исследования показывают, что важно хранить не только информацию о пользователях и сущностях рекомендательной системы, но и предоставлять историю всех взаимодействий пользователя для более успешного конечного результата. Благодаря хранению и первого, и второго, в модели можно будет передать двойной эмбеддинг и о том, и о другом, получив куда более высокие показатели точности [18].

Понимая это, представляется более точная модель данных, хранящая не только свойства сущностей и самого вышеупомянутого пользователя, но и исторический контекст всех действий последнего в приложении.

Важной частью построения модели является, соответственно, и построение достаточного количества свойств каждого из предметов, которые можно будет сопоставить со свойствами и изначальными характеристиками пользователя и избежать одной из основных проблем рекомендательных систем — «холодного старта», благодаря набору характеристик, позволяющих начать рекомендовать подходящие объекты с самого начала. Конечно, не стоит забывать и о возможности реализовать алгоритм «многорукого бандита», описанный в анализе [19].

С пониманием вышеупомянутых факторов объекты строятся и мигрируют в SQL с использованием EF Core и по алгоритмам подхода к разработке схемы реляционных баз данных под названием code-first (в переводе с английского — «сначала код»).

Одной из самых важных вещей в базе данных становится не только умелое описание свойств, но и хранение эмбеддингов, а также информации о пользователе, в том числе конфиденциальной. Последнему будет посвящён следующий подраздел, а вот об эмбеддингах стоит поговорить сразу, так как PostgreSQL предоставляет возможность не только хранить эмбеддинги колонок, но и обеспечивает полнотекстовый поиск по ним, что ускоряет формирование рекомендаций при их формировании в реальном времени. Это обеспечивается с помощью встроенного типа *tsvector* и указывается в конфигурации ORM следующим образом:

```
builder.HasGeneratedTsVectorColumn(w => w.Embeddings,
          "english",
          work => new { Name = work.NameEn, work.DescriptionEn })
      .HasIndex(w => w.Embeddings)
      .HasMethod("GIN");
```

Важно указать формируемый индекс для полнотекстового поиска, всего в PostgreSQL их два: GIN и GIST. Их краткое сравнение приведено в таблице 3.1:

Характеристика	**GIN**	**GiST**
Структура	Инвертированный индекс: ключ — термин, значение — список документов	Обобщённое B-дерево: ключи иерархически структурированы
Скорость вставки и обновления	Медленная вставка и обновление	Быстрая вставка и обновление
Скорость поиска	Очень высокая, особенно при множественных терминах	Высокая, но уступает GIN в полнотекстовом поиске
Используемое пространство	Требует больше дискового пространства	Эффективнее по памяти
Типичные задачи	Полнотекстовый поиск по tsvector, быстрое извлечение документов	Геопоиск, деревья расстояний, поиск ближайших соседей
Идеален для	Преобладание чтения над записью, статичные или редко обновляемые данные	Часто обновляемые данные, гибкие условия поиска

Таблица 3.1 – Сравнение индексов GIN и GiST в PostgreSQL

Так как в данном случае свойства объектов будут обновляться относительно редко (раз в несколько суток), то предпочтительнее использовать ускоряющий поиск по леммам GIN. Данный индекс будет хранить пару термин -> список документов, что позволяет быстро находить все документы, содержащие определённый термин[20]. Все это позволит ускорить алгоритмы рекомендаций и нахождения наиболее подходящих вариантов сущностей для пользователя.

3.3.2 Аутентификация пользователя

Для формирования списка рекомендаций необходим механизм подтверждения и установление идентичности. Для аутентификации в приложении используется встроенное решение ASP.NET — ASP.NET Core Identity.

Механизм аутентификации в ASP.NET Core с использованием Identity представляет собой гибкую и мощную систему для управления пользователями, их паролями, профилями, ролями и утверждениями (клеймами). Вместо того чтобы разработчику приходилось самостоятельно реализовывать все аспекты безопасности, Identity предоставляет готовый каркас. В основе этого каркаса лежит *UserManager*, который отвечает за операции, связанные непосредственно с пользователями, такие как их создание, поиск или изменение пароля. Процесс входа и выхода пользователей, включая

проверку учетных данных и управление сессиями, координируется через *SignInManager*. Для хранения информации о пользователях и их ролях Identity использует концепцию хранилищ, обычно представленную UserStore и RoleStore, которые абстрагируют взаимодействие с базой данных, чаще всего через EF Core, в приложении инъекция этого сервиса и настройка хранения данных пользователях в базе данных выглядит следующим образом:

```
builder.Services.AddIdentityApiEndpoints<User>()
.AddEntityFrameworkStores<PhilosophersCatalogueDbContext>();
```

В текущем приложении будет использоваться куки-аутентификация, так как она предоставляет хороший уровень защищённости, при этом реализуемый встроенными методами без сложной настройки.

Когда пользователь осуществляет запрос на сервер, он предоставляет свои учетные данные. SignInManager обращается к UserManager для поиска пользователя и проверки правильности пароля. Если учетные данные верны, SignInManager создает специальный защищенный куки-файл, который содержит зашифрованную информацию об аутентифицированном пользователе, включая его идентификатор и утверждения. Этот куки отправляется в браузер пользователя. При последующих запросах к защищенным ресурсам приложения браузер автоматически прикрепляет этот куки. Специальный промежуточный слой аутентификации в ASP.NET Core перехватывает этот куки, расшифровывает его и восстанавливает информацию о пользователе, делая ее доступной через HttpContext.User. Таким образом, приложение может легко проверить, аутентифицирован ли пользователь и какими правами он обладает, просто обратившись к этому свойству[21].

Система Identity также заботится о безопасном хранении паролей, используя хеширование с солью, и предоставляет механизмы для защиты от распространенных атак, таких как перебор паролей или межсайтовая подделка запросов (XSRF), если настроена соответствующим образом. Подробный механизм аутентификации изображен на диаграмме 3.1.

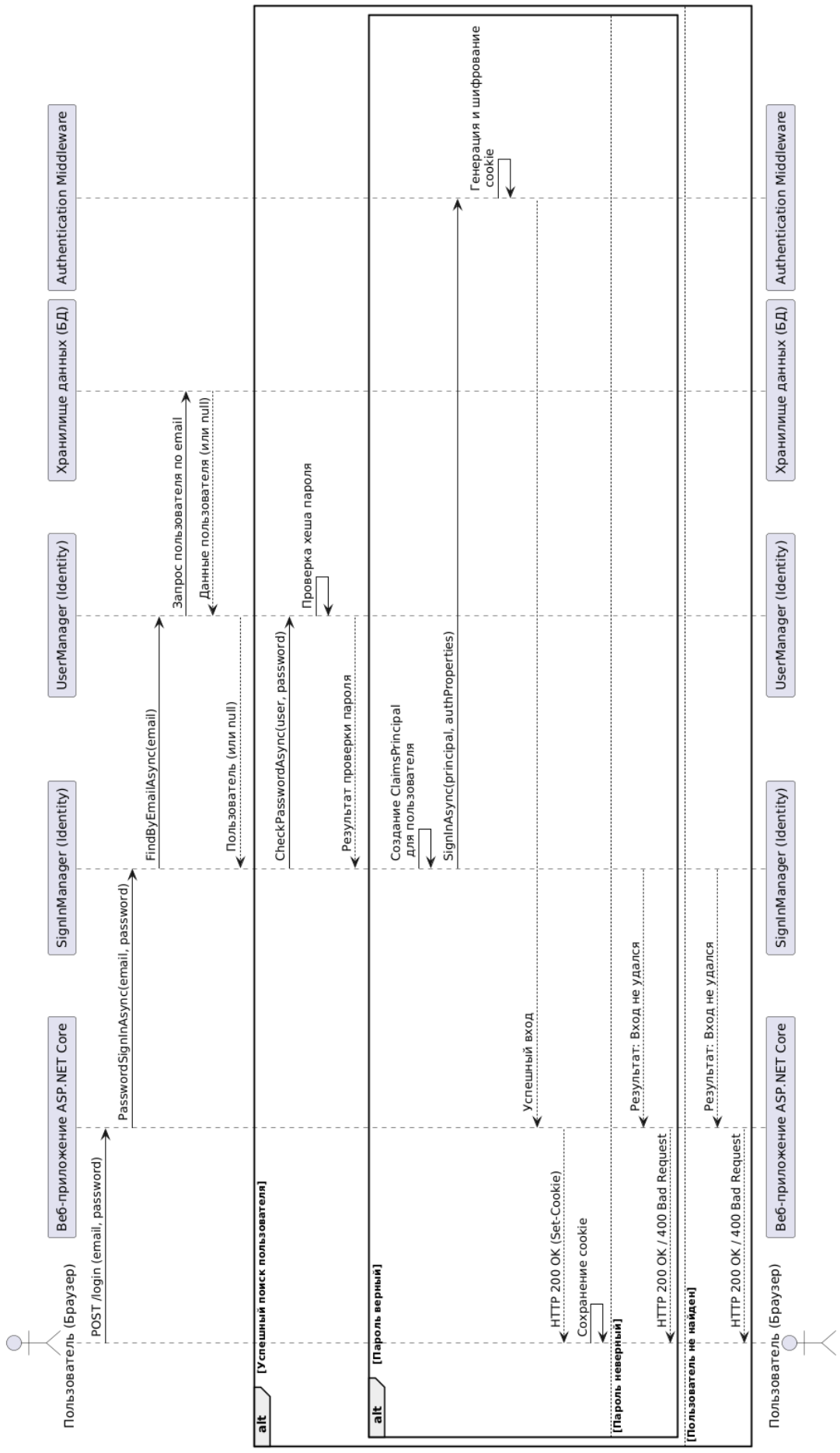


Рисунок 3.1 – Процесс аутентификации и получения куки

3.3.3 Поддержание актуальности информации

Одна из самых важных метрик качества будущего приложения — его способность поддерживать актуальность и точность данных без необходимости постоянной модерации контента. Для этого на базе ASP.NET хоста следует создать регулярно выполняющуюся работу по парсингу обновлённых данных с открытого источника. На начальном этапе разработки таким источником выбрана Википедия, как открытый, актуальный и доступный источник, распространяющийся под свободной лицензией.

Каждые двое суток в приложении будет выполняться работа по обновлению данных из Википедии, создавая новые сущности, обновляя связи и уже имеющиеся элементы с помощью обращения к API Mediawiki. Это достигнуто с помощью библиотеки *Quartz*, которая позволяет установить точное расписание для фоновых заданий.

Листинг основного кода этого фонового задания:

```
private static readonly ImmutableArray<string> Categories =
   [
       "Философы", "Философская литература", "Философские направления", "
          Философские термины", "Философские теории", "Философские
          выражения"
   ];

   public async Task Execute(IJobExecutionContext context)
   {
       logger.LogInformation("WikipediaApiParser: Job Execution started.");

       await BuildInitialPageListAsync(context);

       if (wikipediaPageQueryInfoList.Count != 0)
           await FetchAndStorePageDetailsAsync(context);
   }
```

Вся фоновая работа разделена на два метода — первый, *BuildInitialPageListAsync*, проходит по перечисленным в списке категориям и формирует список идентификаторов страницы в Википедии, второй, *FetchAndStorePageDetailsAsync*, в случае успешного сбора идентификаторов, по ним собирает более подробные данные со страницы и заполняет ими базу данных, постоянно поддерживая актуальность текущих данных в таблице, а также дополняя свойства объекта, расширяя точность рекомендаций.

3.4 Формирование рекомендаций в сервисе

Процесс получения рекомендаций можно разделить на два этапа:

1. При старте сервиса на Python происходит получение данных из базы и генерация эмбеддингов (векторных представлений) для всех элемен-

тов с помощью BERT-трансформера, модель которого загружается с сайта *Hugging Face* перед запуском.

2. При обращении пользователя по адресу сервиса происходит нахождение эмбеддинга для элемента, по которому запрашивается рекомендация и расчет схожести этого эмбеддинга с эмбеддингами всех остальных элементов того же типа. В итоге пользователю возвращаются наиболее похожие элементы.

Этот пример работает для одного из ресурсов, который рассчитывает рекомендации на основе одного присланного элемента (в основном последнего, которому пользователь поставил лайк).

Код формирования эмбеддингов с помощью трансформера из BERT:

```
def prepare_embeddings():
    global BERT_MODEL, EMBEDDING_MATRICES

    print(f"Loading model SentenceTransformer: {MODEL_NAME}...")
    BERT_MODEL = SentenceTransformer(MODEL_NAME)
    print("Модель is loaded.")

    for item_type, items_list in DATA.items():
        if not items_list:
            print(f"Warning: No data for entity type '{item_type}',
                embeddings will not be created.")
            EMBEDDING_MATRICES[item_type] = None
            continue

        print(f"Generate embeddings for '{item_type}'...")
        corpus = [get_text_for_item(item, item_type) for item in items_list]

        if not any(text.strip() for text in corpus):
            EMBEDDING_MATRICES[item_type] = None
            continue

        try:
            embeddings = BERT_MODEL.encode(corpus, convert_to_tensor=False,
                show_progress_bar=True)
            EMBEDDING_MATRICES[item_type] = np.array(embeddings)
        except Exception as e:
            print(f"Exception generating ebeddings for '{item_type}': {e}")
            EMBEDDING_MATRICES[item_type] = None
```

Во время получения рекомендаций для расчёта сходства между сущностями используется косинусное сходство, которое определяет косинус угла между векторами. Если они совершенно не похожи, то итоговым результатом будет 0, если они совершенно идентичны — 1. После ответ сортируется, исключается сам целевой элемент (конечно, сам с собой он будет иметь сходство 1) и выбирается определённое количество лучших кандидатов, возвращаемых в итоге сервисом на основной сервер.

Пример одного из обработчиков в сервисе, механизм которого описан выше:

```
@app.get("/recommendations/{item_type_slug}/{item_id_str}", response_model=
    RecommendationResponse) async def get_recommendations(item_type_slug:
    str, item_id_str: str, top_n: int = 5):
    embedding_matrix = EMBEDDING_MATRICES.get(item_type_slug)

    item_id = uuid.UUID(item_id_str)

    items_list = DATA[item_type_slug]
    target_item_dict = None
    target_item_index = -1

    for i, item_dict in enumerate(items_list):
        if item_dict["id"] == item_id:
            target_item_dict = item_dict
            target_item_index = i
            break

    target_vector = embedding_matrix[target_item_index].reshape(1, -1)

    cosine_similarities = cosine_similarity(target_vector, embedding_matrix).
        flatten()

    similar_items_indices = cosine_similarities.argsort()[-(top_n + 2)
        :-1][::-1]

    recommendations_dicts = []
    for i in similar_items_indices:
        if i == target_item_index:
            continue
        if len(recommendations_dicts) < top_n:
             recommendations_dicts.append(items_list[i])
        else:
            break

    model_map = {
        "philosophers": Philosopher,
        "works": Work,
        "category_schools": CategorySchool,
        "notions": Notion,
    }
    ItemModel = model_map.get(item_type_slug, BaseItem)

    target_item_pydantic = ItemModel(**target_item_dict)
    recommendations_pydantic = [ItemModel(**rec_dict) for rec_dict in
        recommendations_dicts]

    return RecommendationResponse(item=target_item_pydantic, recommendations
        =recommendations_pydantic)
```

Данный сервис является костяком бизнес-логики приложения и выполняет основной функционал, позволяющий отличить последнее от обычного справочника или энциклопедиии.

3.4.1 Разработка пользовательского интерфейса

Интерфейс пользователя стартует с главной страницы приложения, что подразумевает, что именно с неё должна быть начата и разработка клиента для будущего приложения. Главная страница должна быть лаконичной, предоставляя, тем не менее, быстрый доступ ко всем функциям программы, таким как поиск по сущностям и подборка главных рекомендаций для пользователя. Также важную роль в приложении играет хедер, который присутствует не только на стартовой странице, но и по всему приложению, включая логотип и ключевые элементы навигации для основных пользовательских действий, таких как регистрация или вход в аккаунт.

Домашняя страница приложения с данными с сервера и поисковой строкой представлена на рисунке 3.2:

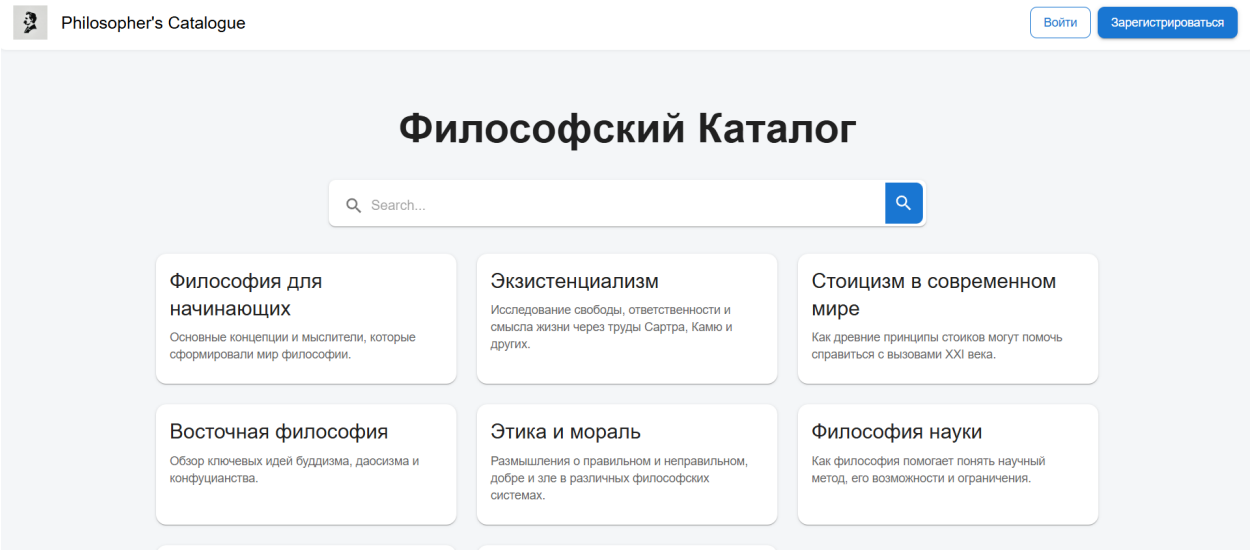


Рисунок 3.2 – Изображение домашней страницы. Видны карточки рекомендуемых пользователю элементов

Страница для входа в систему (изображена на рисунке 3.3) также сделана с уважением к единому стилю, доступному благодаря возможности библиотеки MUI к созданию единой пользовательской темы на всё приложение:

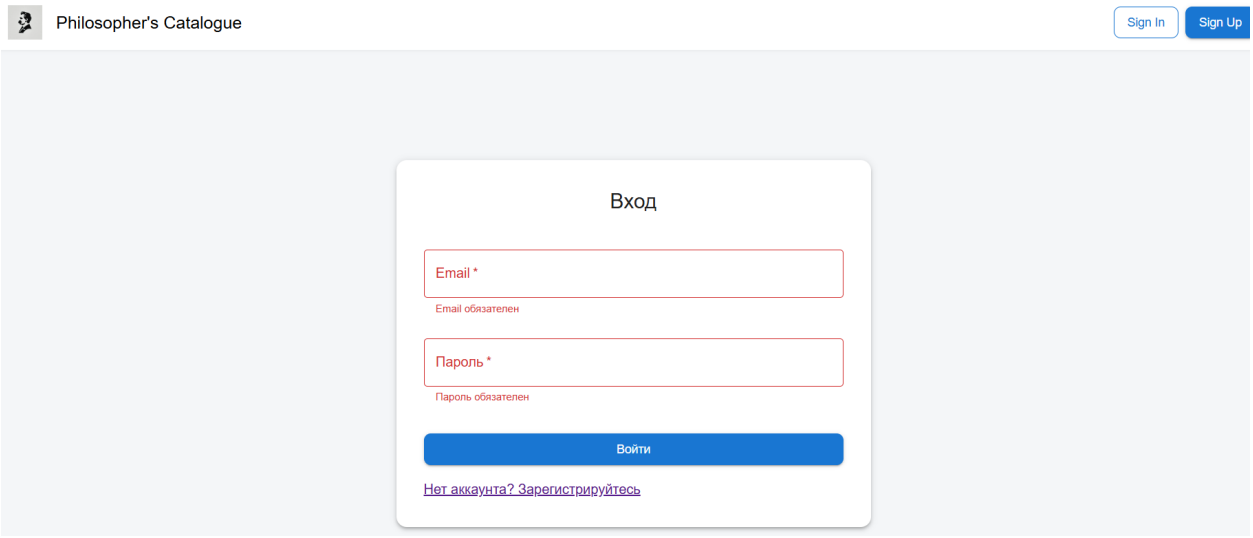


Рисунок 3.3 – Страница входа в систему

При входе в приложение отправляется аутентификационный запрос на интерфейс ASP.NET Identity, который проверяет данные для входа и возвращает результат. В случае положительного результата в браузере сохраняется токен и куки, которые будут использоваться для подтверждения идентичности пользователя, отправляясь вместе с каждым запросом на сервер. Это позволяет получить индивидуальные рекомендации. При отсутствии аутентификации пользователь также будет иметь возможность осуществлять поиск, однако без индивидуальных рекомендаций.

После аутентификации внешне изменения на страницах часто будут не заметны, однако, к примеру, предметы на главной странице, будут отображены исходя из персонализированных рекомендаций.

Важной также становится карточка предмета, которая на начальном этапе содержит заголовок предмета и его описание, а также возможность выставить оценку данного понятия или предмета, при нажатии кнопки «лайк» осуществиться единственная рекомендация, доступная незарегистрированным пользователям: в карусели будут выведены все похожие на текущий предметы. Пример индивидуальной страницы сущности приведён на рисунке 3.4:

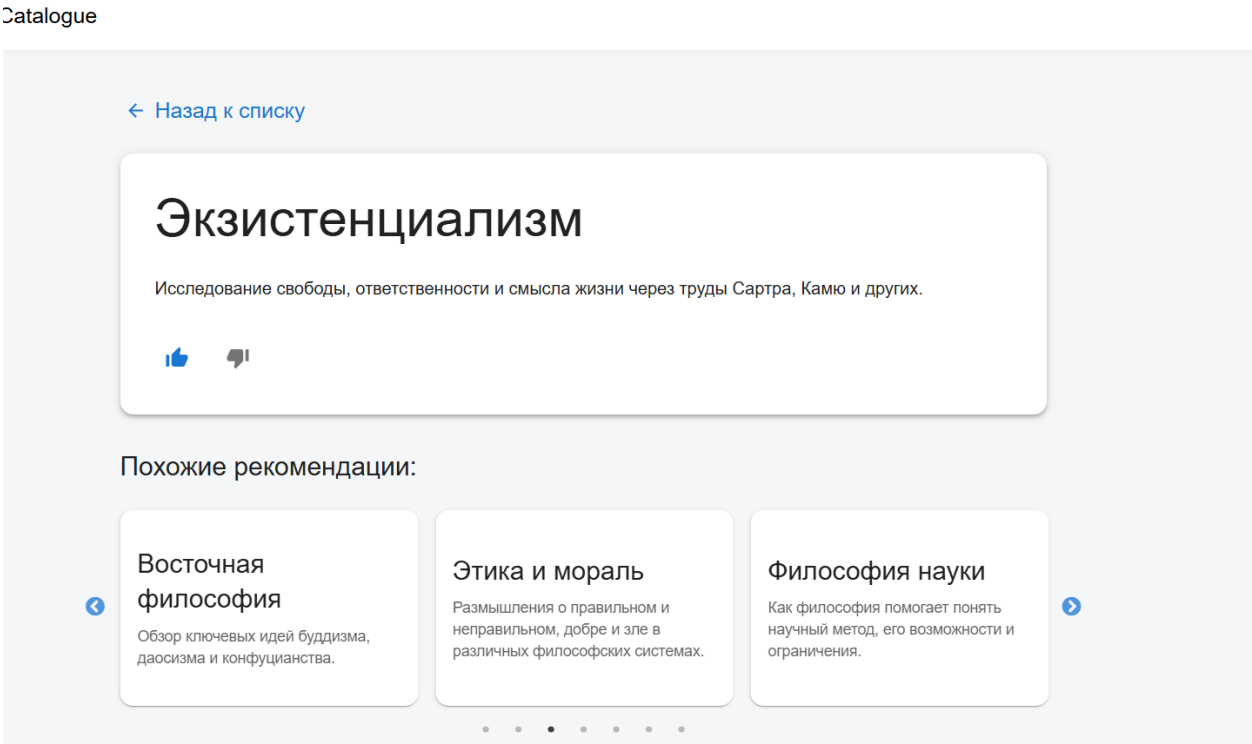


Рисунок 3.4 – Страница предмета после поставленного лайка

При этом поощрением для модели в данном случае будет являться взаимодействие пользователя хотя бы с одной из рекомендаций в списке, что будет отражено в базе данных с помощью посылаемой на сервер после закрытия сущности аналитики по рекомендациям, используя *navigator.sendBeacon()* команду.

3.5 Оценка качества работы

По итогам выполненного дипломного проектирования и окончания разработки приложения было обеспечено несколько важных вещей, упрощающих работу с рекомендательной системой:

– Приложение было контейнеризировано. Каждая из его составляющих: база данных, клиент, сервер и сервис по рекомендациям — все они запускаются в едином образе Docker, позволяя запустить приложение целиком на любом из устройств без какой-либо предварительной настройки;

– был создан удобный интерфейс с аккуратным дизайном для взаимодействия с данными приложения и с рекомендательными сервисами, была использована единая тема приложения, позволяющая сохранить единость и понятно дизайна.

По итогам разработки были написаны юнит-тесты для сервера и стороннего сервиса, а также было проведено ручное тестирование клиента, позволившее провести улучшения, обеспечивающие ещё более качественный пользовательский опыт.

3.6 Вывод

В результате дипломного проектирования была разработана рекомендательная система по философии, которая осуществляет подбор рекомендаций для пользователя с использованием распространённых подходов к построению рекомендательных систем.

Было полностью развёрнуто и контейнерезировано веб-приложение со всеми необходимыми сервисами и ресурсами, в том числе базой данных, сервером и клиентом. База данных была заполнена необходимой для построения рекомендаций информацией, её архитектура была сделана таким образом, чтобы обеспечить наибольшее быстродействие и качество конечного результата. Также был реализован сервис, осуществляющий формирования рекомендаций на основе поступающих данных и обладающий потенциалом переиспользования.

В целом по итогам разработки были учтены и реализованы все поставленные в проектировании цели и требования к разработанному продукту.

4 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СОБСТВЕННЫХ НУЖД

4.1 Характеристика программного средства, разрабатываемого для собственных нужд

Программный модуль рекомендательной системы по философии разрабатывается с целью повышения эффективности восприятия и усвоения философских знаний пользователями, а также для индивидуализации процесса обучения и самообразования в гуманитарной области.

Разрабатываемая система в рамках разрабатываемой её компании ООО «Техартгруп» воспринимается как база для создания коммерческих проектов для учебного заведения или научной организации, а также может продаваться как отдельный продукт в электронные библиотеки и платформы дистанционного обучения.

Целью создания программного модуля рекомендательной системы по философии является автоматизация процесса подбора и предоставления философских материалов, соответствующих интересам, предпочтениям и уровню подготовки пользователя. Система будет использовать алгоритмы машинного обучения и обработки естественного языка для анализа пользовательских запросов и поведения, а также для формирования персонализированных рекомендаций. Данная рекомендательная система будет основой для всех дальнейших разработок компании в данном направлении.

Основными экономическими задачами программного средства являются:

– Упрощение разработки рекомендательных систем в других приложениях компании благодаря созданию готового решения;

– создания продукта, потенциально являющегося товаром для продажи в сторонние компании;

– привлечение научных инвестиций и получение грантов от государства.

Экономическая оценка целесообразности инвестиций в разработку и использование программного средства осуществляется на основе расчета и оценки следующих показателей: чистый дисконтированный доход, рентабельность инвестиций и срок окупаемости инвестиций.

4.2 Расчет инвестиций в разработку программного средства для собственных нужд

Для разработки рекомендательной системы по философии компания-разработчик составила документ. В документе определены требования к разрабатываемому программному средству и его стоимости. Стоимость программного средства рассчитана на основе полных затрат на разработку и включает следующие статьи: основная заработная плата разработчиков, дополнительная заработная плата разработчиков, отчисления на социальные нужды, прочие расходы, общая сумма затрат на разработку.

Расчет затрат на разработку ПО в данном случае будет состоять из следующих пунктов:

– затраты на основную заработную плату разработчиков;

– затраты на дополнительную заработную плату разработчиков;

– отчисления на социальные нужды;

– прочие затраты (амортизационные отчисления, расходы на электроэнергию, командировочные расходы, арендная плата за офисные помещения и оборудование, расходы на управление и реализацию и тому подобное).

4.2.1 Расчет затрат на основную заработную плату разработчикам

Первым этапом расчета затрат на разработку системы является расчет оплаты команды разработчиков. Расчет осуществляется исходя из состава и численности команды, размера месячной заработной платы каждого участника команды, а также трудоемкости работ, выполняемых при разработке программного средства отдельными исполнителями по формуле:

Основная заработная плата исполнителей проекта определяется по формуле:

$$З_О = К_{ПР} \cdot \sum_{i=1}^{n} З_{Чi} \cdot t_i, \tag{4.1}$$

где n – количество исполнителей, занятых разработкой конкретного ПО;

$К_{ПР}$ – коэффициент премий (1,5);

$З_{Чi}$ – часовая заработная плата i-го исполнителя (руб.);

t_i – трудоемкость работ, выполняемых i-м исполнителем (ч).

На 2025 год расчетная норма рабочего времени для пятидневной рабочей недели составляет 168 часов, 8 часов работы в день, среднемесячная расчетная норма рабочего времени – 21 день.

Расчет основной заработной платы представлен в таблице 4.1.

Таблица 4.1 – Расчет затрат на основную заработную плату разработчиков

№	Участник команды	Вид выполняемой работы	Месячная заработная плата, руб.	Часовая заработная плата, руб.	Трудоемкость работ, ч.	Зарплата по тарифу, руб.
1	2	3	4	5	6	7
1	Руководитель проекта	аналитика, разработка архитектуры	9000,00	53,57	40	2142,85
2	Инженер-программист	разработка	6000	35,7	400	14 285,71
3	Тестировщик	тестирование	3500	20,83	40	833.3
Премия и иные выплаты (20%)						4960,31
Итого затраты на основную заработную плату разработчиков						29761,87

4.2.2 Расчёт затрат на дополнительную заработную плату разработчикам

Затраты на дополнительную зарплату разработчиков включают выплаты, предусмотренные законодательством о труде (оплата трудовых отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяются по формуле:

$$З_д = \frac{З_o \cdot H_д}{100}, \tag{4.2}$$

где $H_д$ – норматив дополнительной заработной платы(20 %);
$З_o$ – затраты на основную заработную плату (руб.);
Дополнительная заработная плата составит:

$$З_д = \frac{29761,872 \cdot 20}{100} = 5952,37 \text{ руб.}$$

4.2.3 Расчёт отчислений на социальные нужды

В расчете отчислений на социальные нужды отражаются обязательные отчисления по установленным законодательством тарифам в фонд социальной защиты населения, а также расходы предприятия на обязательное социальное медицинское страхование некоторых категорий работников в соответствии с законодательством. Расчет размера отчислений в фонд

социальной защиты населения и на обязательное страхование определяется в соответствии с действующими законодательными актами Республики Беларусь и вычисляется по формуле:

$$\text{Р}_{\text{соц}} = \frac{(\text{З}_o + \text{З}_\text{д}) \cdot H_{\text{соц}}}{100}, \tag{4.3}$$

где $H_{\text{соц}}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34,6%, где 34% идут в фонд социальной защиты и 0,6% на обязательное страхование).

$$\text{З}_{\text{сх}} = \frac{(29761,872 + 5952,3744) \cdot 34,6}{100} = 12357,13 \text{ руб.}$$

Согласно расчетам, размер отчислений в фонд социальной защиты и на обязательное страхование составляет 12357,13 рублей.

4.2.4 Расчет затрат на прочие расходы

Прочие расходы связаны с функционированием организации-разработчика в целом, например: затраты на аренду офисных помещений, отопление, освещение, амортизацию основных производственных фондов и так далее. При расчете данной статьи затрат учитывается норматив прочих затрат в целом по организации. В данном случае норматив прочих затрат равен 20%. Размер затрат на прочие расходы рассчитывается по формуле:

$$\text{З}_{\text{пз}} = \frac{\text{З}_o \cdot \text{Н}_{\text{пз}}}{100}, \tag{4.4}$$

где $\text{Н}_{\text{пз}}$ – норматив прочих затрат (20 %).

Подставим значение из выражения в формулу 4.4 и произведём расчёт $\text{Р}_{\text{пр}}$:

$$\text{З}_{\text{пз}} = \frac{29761,872 \cdot 20}{100} = 5952,37 \text{ руб.}$$

4.2.5 Расчёт суммы затрат на разработку

Полная сумма затрат на разработку программного обеспечения находится путем суммирования всех рассчитанных статей затрат.

$$\text{З}_\text{р} = \text{З}_\text{о} + \text{З}_\text{д} + \text{Р}_{\text{соц}} + \text{Р}_{\text{пр}}, \tag{4.5}$$

Подставим результаты вычислений в формулу 4.5 и произведем расчет $\text{З}_\text{р}$:

$$\text{З}_\text{р} = 29761{,}872 + 5952{,}37 + 12357{,}13 + 5952{,}37 = 54023{,}74 \text{ р.} \tag{4.6}$$

Согласно расчетам, сумма затрат на разработку составляет 54023,74 рубля.

Таблица 4.2 – Затраты на разработку программного обеспечения

Статья затрат	Сумма, руб.
Основная заработная плата команды разработчиков	29761,87
Дополнительная заработная плата команды разработчиков	5952,37
Отчисления в фонд социальной защиты и обязательного страхования	12357,13
Прочие затраты	5952,37
Общая сумма затрат на разработку	54023,74

4.3 Расчет экономического эффекта от использования программного средства для собственных нужд

Экономия на заработной плате и начислениях на заработную плату сотрудников за счёт снижения трудоёмкости работ необходимо рассчитывать по формуле 4.7:

$$Э_{з.п} = K_{пр} \cdot \left(t_{р}^{без\ п.с} - t_{р}^{с\ п.с}\right) \cdot T_{ч} \cdot N_{п} \cdot \left(1 + \frac{H_{д}}{100}\right) \cdot \left(1 + \frac{H_{соц}}{100}\right) \quad (4.7)$$

где $K_{пр}$ — коэффициент премий (по фактическим данным предприятия или в диапазоне 1,5–2); $t_{р}^{без\ п.с}$, $t_{р}^{с\ п.с}$ — трудоёмкость выполнения работ сотрудниками до и после внедрения программного средства, ч; $T_{ч}$ — часовой оклад (часовая тарифная ставка) сотрудника, использующего программное средство, р.; $N_{п}$ — плановый объём работ, выполняемый сотрудниками; $H_{д}$ — норматив дополнительной заработной платы (10–20 %); $H_{соц}$ — ставка отчислений от заработной платы, включаемых в себестоимость, (34,6 %).

$$Э_{з.п} = 1{,}2 \cdot (3 - 2) \cdot 15 \cdot 2016 \cdot \left(1 + \frac{20}{100}\right) \cdot \left(1 + \frac{34{,}6}{100}\right) = 58612{,}37 \text{ р.} \quad (4.8)$$

Экономия на заработной плате и начислениях на заработную плату в результате сокращения численности работников рассчитывается по формуле 4.9:

$$Э_{з.п}^{п} = \sum_{i=1}^{n} \Delta Ч_{i} \cdot З_{i} \cdot \left(1 + \frac{H_{д}}{100}\right) \cdot \left(1 + \frac{H_{соц}}{100}\right) \quad (4.9)$$

где n — категории работников, высвобождаемых в результате внедрения программного средства; $\Delta Ч_i$ — численность работников i-й категории, высвобожденных после внедрения программного средства, чел.; $З_i$ — годовая заработная плата высвобождённых работников i-й категории после внедрения программного средства, р.; $H_{соц}$ — норматив отчислений от заработной платы в соответствии с законодательством, %.

Экономия на заработной плате и начислениях на заработную плату в результате сокращения численности работников составляет 0 р., поскольку сотрудники не были уволены.

Экономия на материальных ресурсах рассчитывается по формуле 4.10:

$$Э_м = K_{т.р} \cdot \left(H_р^{без\ п.с} - H_р^{с\ п.с}\right) \cdot C_м \cdot N_п \tag{4.10}$$

где $K_{т.р}$ — коэффициент транспортных расходов (по данным предприятия или 1,05–1,2); $H_{рс}^{без\ п.с}$, $H_{рс}^{с\ п.с}$ — норма расхода материальных ресурсов при выполнении работ сотрудниками до и после внедрения программного средства, нат. ед.; $C_м$ — цена за единицу материального ресурса, р.; $N_п$ — плановый объём работ, выполняемых сотрудниками с использованием программного средства.

$$_м = 1{,}05 \cdot (5 - 5{,}5) \cdot 10 \cdot 142 = 745{,}5 \text{ р.}$$

Экономия на материальных ресурсах равна 745,5 р.

Экономическим эффектом при использовании программного средства является прирост чистой прибыли, полученной за счёт экономии на текущих затратах предприятия, который рассчитывается по формуле 4.11:

$$\Delta П_ч = (Э_{тек} - \Delta З_{тек}^{п.с}) \left(1 - \frac{H_п}{100}\right) \tag{4.11}$$

где $Э_{тек}$ — экономия на текущих затратах при использовании программного средства, р.; $\Delta З_{тек}^{п.с}$ — прирост текущих затрат, связанных с использованием программного средства (затраты на сопровождение программного средства, затраты на интернет-трафик и т. п.), р.; $H_п$ — ставка налога на прибыль согласно действующему законодательству 20%.

$$\Delta П_ч = ((58612{,}37 + 745{,}5) - 0) \cdot \left(1 - \frac{20\%}{100}\right) = 47486{,}29 \text{ р.}$$

4.4 Расчет показателей экономической эффективности разработки и использования программного средства в организации

Оценка экономической эффективности разработки и использования программного средства для собственных нужд зависит от результата сравнения затрат на его разработку (модернизацию, совершенствование) и полученного экономического эффекта (годового прироста чистой прибыли).

Так как сумма инвестиций (затрат) осуществляется более чем за год, то для организации рассчитывается несколько показателей экономической эффективности. Для приведения доходов и затрат к настоящему моменту времени определяется коэффициент дисконтирования по формуле 4.12:

$$\alpha_t = \frac{1}{(1 + d_{\text{н}})^{t - t_{\text{п}}}} \tag{4.12}$$

где d – норма дисконта (в долях единиц), равная или больше средней процентной ставки по банковским депозитам, действующей на момент осуществления расчётов;

t – порядковый номер года периода реализации инвестиционного проекта (предполагаемый период использования разрабатываемого ПО пользователем и время на разработку); t_p – расчётный год, к которому приходятся доходы и инвестиционные затраты ($t_p = 1$).

Норму дисконта принимаем равным ставке рефинансирования Национального банка Республики Беларусь – 9,5%. Расчетный период составит четыре года. Таким образом, коэффициенты дисконтирования за каждый год составляют:

$$\alpha_1 = \frac{1}{(1 + 0,095)^{1-1}} = 1.$$

$$\alpha_2 = \frac{1}{(1 + 0,095)^{2-1}} = 0,91.$$

$$\alpha_3 = \frac{1}{(1 + 0,095)^{3-1}} = 0,83.$$

$$\alpha_4 = \frac{1}{(1 + 0,095)^{4-1}} = 0,76.$$

В течение первого года осуществляется разработка приложения, поэтому в первый год экономический эффект будет меньше планируемого. Для того, чтобы учесть этот факт, необходимо выяснить, сколько времени будет затрачено на разработку приложения. Так как работа команды

разработчиков осуществляется поэтапно, то затраченное время будет равно сумме трудоемкости работ команды и составит 480 часов.

Таблица 4.3 – Расчёт эффективности инвестиций (затрат) в реализацию проектного решения

Показатель	**1-й год**	**2-й год**	**3-й год**	**4-й год**
1. Прирост чистой прибыли, р.	21210,54	47486,29	47486,29	47486,29
2. Дисконтированный результат, р.	21210,54	43212,52	39413,62	36089,58
3. Инвестиции (затраты) в реализацию проектного решения, р.	54023,74	0	0	0
4. Дисконтированные инвестиции, р.	54023,74	0	0	0
5. Чистый дисконтированный доход по годам	-32813,2	43212,52	39413,62	36089,58
6. Чистый дисконтированный доход нарастающим итогом, р.	-32813,2	10399,32	49812,94	85902,52
7. Коэффициент дисконтирования, доли единицы	1,0	0,91	0,83	0,76

В данном случае дисконтированный эффект нарастающим итогом превысит дисконтированные инвестиции на третий год. Дисконтированный срок окупаемости рассчитывается по формуле:

$$T_{\text{ок}}(PP) = \frac{\sum\limits_{t=1}^{n} t}{\frac{1}{n} \cdot \sum\limits_{t=1}^{n} \Delta\Pi_t},$$

Таким образом, дисконтированный срок окупаемости равен:

$$T_{\text{ок}}(PP) = \frac{54023{,}74}{\frac{1}{4}(21210{,}54 + 47486{,}29 + 47486{,}29 + 47486{,}29)} = 1{,}82 \text{ года}$$

Индекс доходности инвестиций рассчитывается по формуле:

$$\text{ИД(PI)} = \frac{\sum\limits_{t=1}^{n} \Delta\Pi_t \cdot \alpha_t}{\sum\limits_{t=1}^{n} З_t \cdot \alpha_t},$$

Таким образом, индекс доходности инвестиций равен

$$\text{ИД(PI)} = \frac{21210{,}54 + 43212{,}52 + 39413{,}62 + 36089{,}58}{54023{,}74} = 2{,}59$$

В результате экономического обоснования разработки рекомендательной системы по философии видно, что разработка данного программного средства является эффективной, и вложение инвестиций в разработку целесообразно.

4.5 Вывод

Разработка рекомендательной системы по философии направлена на создание готовой библиотеки и программного продукта для ООО «Техартугруп», который будет использоваться как для предоставления за отдельную плату в научные университеты и другие компании, так и для разработки своих приложений как готовое решение. Внедрение данного программного средства позволит сократить трудозатраты на разработку будущих приложений компаний, ускорить выполнение типовых инфраструктурных задач, что в совокупности повысит качество предоставляемых услуг.

Расчет инвестиций в разработку показал, что основная заработная плата разработчиков составит 29761,87 руб., а общая сумма затрат с учетом дополнительных выплат, отчислений на социальные нужды и прочих расходов достигнет 54023,74 руб. Экономический эффект от использования системы, рассчитанный на основе экономии текущих затрат и прироста чистой прибыли, составил 47486,29 руб., что подтверждает целесообразность инвестиций.

По итогам экономического обоснования были получены следующие результаты:

1. Общая стоимость затрат на разработку системы верификации резервного копирования серверов в среде виртуализации составила 58335,38 рублей.

2. Прирост чистой прибыли от внедрения программного средства достиг 47486,29 рублей.

3. Инвестиции окупятся за 1,82 года (примерно 664 дня), а индекс доходности инвестиций составил 2,59, что свидетельствует об эффективности проекта.

ЗАКЛЮЧЕНИЕ

По итогам проделанного дипломного проекта были проанализированы множественные подходы к созданию рекомендательных систем, существующие решения и наиболее популярные подходы к разработке, используемые в крупных коммерческих приложениях, по итогам анализа были сделаны выводы о лучших архитектурах и подходах для такой предметной области, как философия.

После анализа на основе полученных выводов было выполнено проектирование, в котором была выбрана архитектура приложения, а также разработаны различные диаграммы, отражающие приложения и дающие понимание по шагах разработки.

Наконец, на основе проектирования и выставленных целей было разработано и развёрнуто в контейнере микросервисное веб-приложение, представляющее собой рекомендательную систему по философии, призванную формированием индивидуальных рекомендаций привлечь пользователя к этому роду интеллектуальной деятельности и помочь разобраться с его непростыми терминами и категориями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Rafael Glauber, Angelo Loula. Collaborative Filtering vs. Content-Based Filtering: differences and similarities / Angelo Loula Rafael Glauber // Information Retrieval. — 2019. — Режим доступа: https://arxiv.org/abs/1912.08932. — Дата доступа: 24.04.2025.

[2] Ferrari Dacrema Maurizio Cremonesi Paolo, Jannach Dietmar. "Are we really making much progress? A worrying analysis of recent neural recommendation approaches-/ Jannach Dietmar Ferrari Dacrema Maurizio, Cremonesi Paolo. — Proceedings of the 13th ACM Conference on Recommender Systems. ACM, 2019. — 110 P. — Режим доступа: https://dl.acm.org/doi/10.1145/3298689.3347058 – Дата доступа: 06.03.2025.

[3] A Multi-Armed Bandit Model Selection for Cold-Start User Recommendation / Crícia Z. Felício, Klérisson V.R. Paixão, Celia A.Z. Barcelos, Philippe Preux // Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization (July 7th, 2017). — Bratislava: Association for Computing Machinery, 2017. — Pp. 32–33. — Режим доступа: https://dl.acm.org/doi/10.1145/3079628.3079681 – Дата доступа: 06.03.2025.

[4] Blanda, Stephanie. Online Recommender Systems – How Does a Website Know What I Want? [Electronic resource]. — Mode of access: http://blogs.ams.org/mathgradblog/2015/05/25/online-recommender-systems-website-want/. — Date of access: 25.02.2025.

[5] Gomez-Uribe, Carlos A. The Netflix Recommender System / Carlos A Gomez-Uribe // ACM Transactions on Management Information Systems. — 2015. — Режим доступа: https://dl.acm.org/doi/10.1145/2843948. — Дата доступа: 24.04.2025.

[6] Ozon. Алгоритм рекомендаций на Ozon [Электронный ресурс]. — Способ доступа: https://docs.ozon.ru/legal/terms-of-use/site/algorithms/recomendation-algorithms/. — Дата доступа: 25.02.2024.

[7] Goodrow, Cristos. On YouTube's recommendation system [Электронный ресурс]. — Mode of access: https://blog.youtube/inside-youtube/on-youtubes-recommendation-system/. — Date of access: 25.02.2025.

[8] Iangwei Pan Gary Tang, Henry Wang. Recommending for Long-Term Member Satisfaction at Netflix [Электронный ресурс]. — Mode of access: https://netflixtechblog.com/recommending-for-long-term-member-satisfaction-at-netflix-ac15cada49ef. — Date of access: 25.02.2025.

[9] Le, Quoc V. Distributed Representations of Sentences and Documents / Quoc V. Le, Tomas Mikolov // CoRR. — 2014. https://cs.stanford.edu/~quocle/paragraph_vector.pdf.

[10] Huang, Ran. Improved content recommendation algorithm integrating semantic information / Ran Huang // Journal of Big Data. — 2023. — Vol. 10, no. 1. — 84 P. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00776-7.

[11] Lops, Pasquale. Content-based Recommender Systems: State of the Art and Trends / Pasquale Lops, Marco de Gemmis, Giovanni Semeraro // Recommender Systems Handbook. — Springer, 2011. — Pp. 73–105.

[12] Gheewala, Shivangi. In-depth survey: deep learning in recommender systems—exploring prediction and ranking models, datasets, feature analysis, and emerging trends / Shivangi Gheewala, Shuxiang Xu, Soonja Yeom // Neural Computing and Applications. — 2025. https://link.springer.com/article/10.1007/s00521-024-10866-z.

[13] IBM. What is content-based filtering? — 2023. — Accessed: 2025-04-30. https://www.ibm.com/think/topics/content-based-filtering.

[14] Brusilovsky, Peter. User Models for Adaptive Hypermedia and Adaptive Educational Systems / Peter Brusilovsky, Eva Millán // The Adaptive Web. — Springer, 2007. — Pp. 3–53.

[15] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova // Computer Science. — 2018. — Режим доступа: https://arxiv.org/abs/1810.04805v2 – Дата доступа: 06.03.2025.

[16] Collective authors. Difference Between MVC, MVP and MVVM Architecture Pattern in Android [Электронный ресурс]. — Режим доступа: https://www.geeksforgeeks.org/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android. — Дата доступа: 14.05.2025.

[17] Microsoft. Microsoft Recommenders Toolkit. — https://github.com/microsoft/recommenders. — 2020. — Accessed: 2025-05-06.

[18] Gongshan He Dongxing Zhao, Lixin Ding. Dual-embedding based Neural Collaborative Filtering for Recommender Systems / Lixin Ding Gongshan He, Dongxing Zhao // School of Computer Science. — 2021. — Режим доступа: https://dl.acm.org/doi/10.1145/2843948. — Дата доступа: 24.04.2025.

[19] Daniel Billsus, Michael J. Pazzani. Content-Based Recommendation Systems / Michael J. Pazzani Daniel Billsus // The Adaptive Web. — Springer, 2007. — Pp. 325–341.

[20] The PostgreSQL Global Development Group. — PostgreSQL Text Search Documentation, 2024. — Accessed: 2025-05-14. https://www.postgresql.org/docs/current/textsearch.html.

[21] Microsoft Docs. Introduction to ASP.NET Core Identity. — https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-9.0&tabs=visual-studio. — 2025. — Accessed: 2025-05-14.

[22] Официальный сайт компании ООО «Техартгруп» [Электронный ресурс]. — Режим доступа: https://itechartgroup.by. — Дата доступа: 24.06.2025.

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Отчет о проверке на заимствования №1

Автор: user@pochta.ru / ID: 6531736
Проверяющий: (user@pochta.ru / ID: 6531736)

Отчет предоставлен сервисом «Антиплагиат»- http://users.antiplagiat.ru

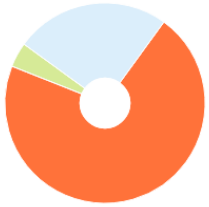
ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 5
Начало загрузки: 12.04.2019 12:39:39
Длительность загрузки: 00:00:08
Имя исходного файла: Документ 1
Размер текста: 62 кБ
Символов в тексте: 32106
Слов в тексте: 3795
Число предложений: 194

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 12.04.2019 12:39:47
Длительность проверки: 00:00:01
Комментарии: не указано
Модули поиска: Модуль поиска Интернет, Цитирование

ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
70,6%	4,44%	24,96%



ПОДОЗРИТЕЛЬНЫЙ ДОКУМЕНТ

Есть подозрения на следующие группы обходов: **СОКРЫТИЕ** на страницах:1, 7, 8, 9, 10, 11, 12, 13, 14, 15... еще на 2 стр.; **ВСТАВКА** на страницах:7, 8, 9, 10, 11, 12, 13, 14, 15, 16... еще на 1 стр.

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся

Обозначение	*Наименование*	*Дополнительные сведения*
	Текстовые документы	
БГУИР ДП 1-40 03 01 ХХХ ПЗ	Пояснительная записка	68 с.
	Отзыв руководителя	
	Рецензия	
	Справка о внедрении результатов	
	Отчет о проверке дипломного проекта на заимствования	
	Графический материал	
ГУИР.17001-01 90 01-1		Формат А1
ГУИР.17001-01 91 01-1		Формат А1
ГУИР.17001-01 92 01-1		Формат А1
ГУИР.17001-01 93 01-1		Формат А1
ГУИР.17001-01 94 01-1		Формат А1
ГУИР.17001-01 95 01-1		Формат А1

					БГУИР ДП 1- 40 03 01 ХХХ Д1	
Изм.	*Л.*	*№ докум.*	*Подп.*	*Дата*	*Система программного управления фрезерованием* Ведомость документов	*Лит* / *Лист* / *Листов*: Т / 68 / 68
Разраб.		*Студент*				
Пров.		*Руководитель*				*Кафедра ИИТ гр. ххххх*
Т.контр.		*Консультант*				
Н.контр.		*Захаров*				
Утв.		*Шункевич*				