

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г.
ШУХОВА» (БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Курсовой проект

по дисциплине: “Конструирование программного обеспечения”
тема: “Разработка адаптивной системы рекомендаций для интернет-
магазина”

Автор работы _____ Игнатъев Артур Олегович ПВ-223
(подпись)

Руководитель проекта _____ Заикин Александр Олегович
(подпись)

Оценка _____

Белгород 2025 г.

Содержание

Введение.....	3
1. Анализ предметной области и создание базовой структуры программного проекта.....	4
1.1 Анализ предметной области.....	4
1.2 Требования к системе.....	5
1.3 Базовая структура проекта.....	6
2. Декомпозиция и визуализация программного проекта.....	8
2.1 Диаграмма зависимостей задач.....	8
2.2 Схема взаимодействия.....	9
3. Методы управления.....	10
3.1 Выбор и обоснование методологии.....	10
4. Организация рабочего пространства управления проектом.....	12
4.1 Принципы организации рабочего пространства.....	12
4.2 Инструменты организации рабочего пространства.....	12
5. Система контроля и мониторинга реализации проекта.....	14
5.1 Философия и принципы системы контроля.....	14
6. Оценка затрат на реализацию и внедрение.....	15
6.1 Оценка затрат на реализацию.....	15
6.2 Оценка затрат на внедрение.....	16
7. Анализ рисков и меры по их предотвращению.....	17
7.1 Технические риски.....	17
7.2 Внешние риски.....	18
Заключение.....	20
Список источников и литературы.....	21

Введение

Данный курсовой проект посвящён созданию комплексного плана проекта по реализации и внедрению адаптивной системы рекомендаций для интернет-магазина. Система предназначена для персонализации предложений товаров на основе поведения пользователей: для авторизованных с учётом истории просмотров и покупок, популярных товаров, ценового сегмента, дня зарплаты и средних трат; для неавторизованных на основе популярных товаров и данных о просмотрах, хранимых в cookie.

Целью проекта является приобретение практических навыков создания и управления программными проектами в области конструирования программного обеспечения.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ предметной области и создать базовую структуру программного проекта.
2. Провести декомпозицию и визуализацию программного проекта.
3. Выбрать и обосновать методологию управления проектом.
4. Организовать рабочее пространство управления проектом.
5. Определить систему контроля и мониторинга реализации проекта.
6. Оценить затраты на реализацию и внедрение.
7. Провести анализ рисков и разработать меры по их предотвращению.

1. Анализ предметной области и создание базовой структуры программного проекта

1.1 Анализ предметной области

Адаптивные системы рекомендаций товаров стали неотъемлемым элементом современных интернет-магазинов и маркетплейсов. Лидеры рынка активно используют подобные системы, что позволяет значительно повышать конверсию, средний чек и лояльность пользователей. По данным исследований, персонализированные рекомендации могут увеличивать продажи на 10–35%, а также повышать время пребывания пользователя на сайте и частоту повторных покупок.

Ключевыми преимуществами адаптивных систем рекомендаций являются: персонализация предложений на основе поведения пользователя, увеличение вероятности покупки за счёт релевантных подсказок, поддержка как авторизованных, так и неавторизованных пользователей, адаптация к контексту.

1.2 Требования к системе

Функциональные требования:

1. Сбор и анализ данных о поведении авторизованных пользователей: история просмотров, покупки, средний чек, ценовой сегмент, дата зарплат.
2. Генерация рекомендаций на основе комбинации факторов: просмотренные товары, купленные товары, популярные товары в категории, соответствие бюджету и контексту.
3. Для неавторизованных пользователей: отображение популярных товаров по умолчанию и персонализация на основе данных о просмотрах, хранимых в cookie.
4. Интеграция с существующей инфраструктурой интернет-магазина.
5. Выдача рекомендаций в виде списка карточек товаров.
6. Обновление рекомендаций в реальном времени при взаимодействии пользователя с сайтом.

Нефункциональные требования:

1. Высокая производительность: время отклика рекомендаций не более 200 мс.
2. Масштабируемость: поддержка тысяч одновременных запросов в пиковые нагрузки.
3. Точность и релевантность рекомендаций.
4. Надёжность и отказоустойчивость.

1.3. Базовая структура проекта

В качестве архитектуры выбрано микросервисное решение. Такой подход обеспечивает гибкость разработки, независимое масштабирование отдельных компонентов, удобство тестирования и развёртывания, а также отказоустойчивость системы в целом.

Система будет состоять из четырёх основных микросервисов:

1. Recommendation API Gateway – единая точка входа для запросов от фронтенда интернет-магазина. Отвечает за аутентификацию пользователей, обработку cookie для неавторизованных сессий, маршрутизацию запросов к другим сервисам и формирование финального ответа.
2. User Behavior Service – сервис сбора, хранения и предоставления данных о поведении пользователей.
3. Recommendation Engine – основной сервис расчёта рекомендаций. Реализует логику алгоритма: комбинацию коллаборативной и контентной фильтрации, rule-based правила, ранжирование и формирование персонализированного списка товаров.
4. Popularity & Analytics Service – сервис расчёта популярных товаров, агрегации статистики по категориям, ценовым сегментам и общей аналитики, необходимой для базовых и fallback-рекомендаций.

Для сервисов с высокой нагрузкой на запросы выбран язык программирования Go. Он обеспечивает высокую производительность, эффективную работу с параллелизмом, низкое потребление памяти и быстрое время отклика, что критично для обработки большого количества одновременных запросов. Go имеет отличную стандартную библиотеку для работы с HTTP, gRPC и базами данных.

Основные модули этих сервисов:

1. HTTP/gRPC-сервер.

2. Модуль аутентификации и работы с cookie.
3. Клиенты для взаимодействия с другими микросервисами и базой данных.

Для Recommendation Engine и Popularity & Analytics Service выбран язык программирования Python. Он обладает широкой экосистемой библиотек для работы с данными и реализации алгоритмов рекомендаций, что значительно упрощает и ускоряет разработку сложной логики рекомендаций.

Основные модули этих сервисов:

1. Модуль обработки и анализа истории пользователя.
2. Реализация алгоритмов рекомендаций.
3. Модуль агрегации популярности и аналитики.

Также в проекте используются общие компоненты инфраструктуры:

1. Контейнеризация: Docker - для упаковки каждого микросервиса и его зависимостей.
2. Оркестрация в разработке: Docker Compose - для локального запуска и тестирования всей системы.
3. Базы данных:
 - PostgreSQL - для хранения структурированных данных.
 - Redis - для кэширования часто запрашиваемых рекомендаций, хранения сессионных данных и быстрых операций.

2. Декомпозиция и визуализация программного проекта

2.1 Диаграмма зависимостей задач

Для эффективного управления проектом и минимизации простоев в работе команды необходимо четко определить задачи и логические связи между задачами. Диаграмма зависимостей задач визуализирует эти отношения, показывая, какие задачи должны быть выполнены перед началом других и какие могут выполняться параллельно.

На представленной ниже диаграмме отражены ключевые зависимости между основными этапами разработки адаптивной системы рекомендаций для интернет-магазина:

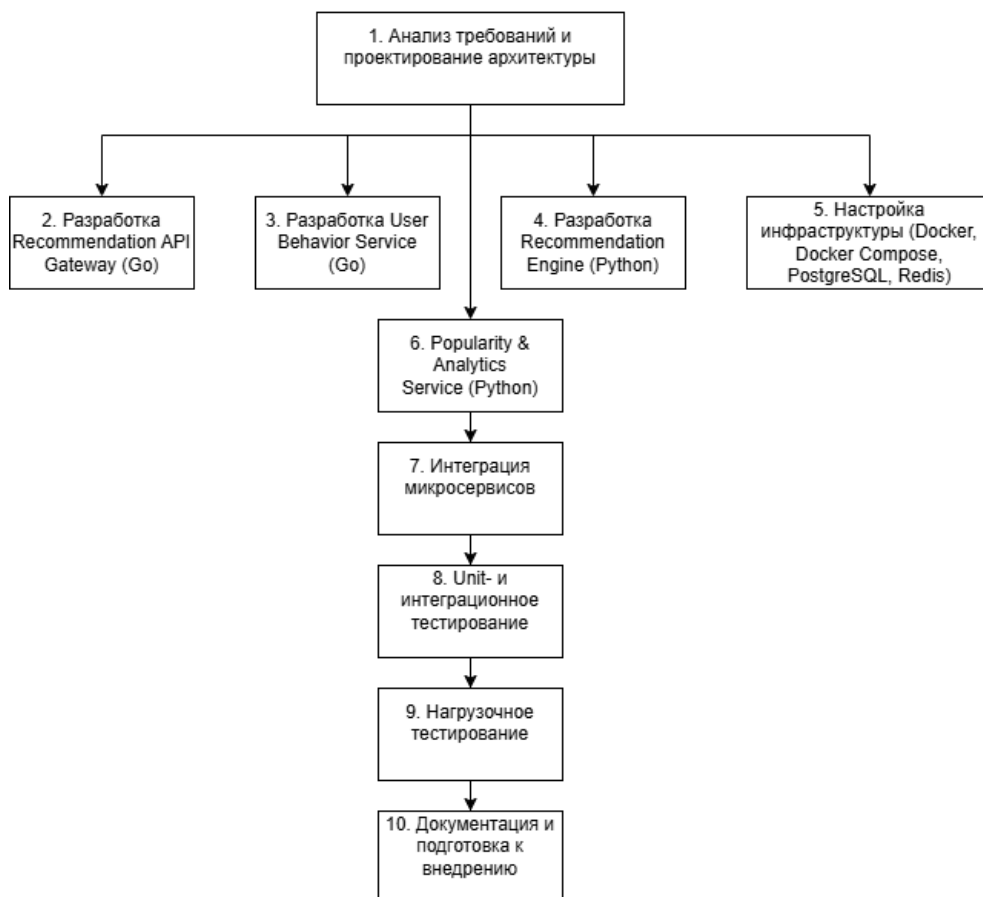


Рисунок 1. Диаграмма зависимостей задач

2.2. Схема взаимодействия микросервисов

Для понимания и обеспечения эффективной коммуникации между компонентами системы необходима визуализация архитектуры взаимодействия микросервисов. Представленная ниже схема демонстрирует, как различные компоненты системы обмениваются информацией, какие протоколы используются для коммуникации и как организованы потоки запросов от клиента до обработки и обратно.

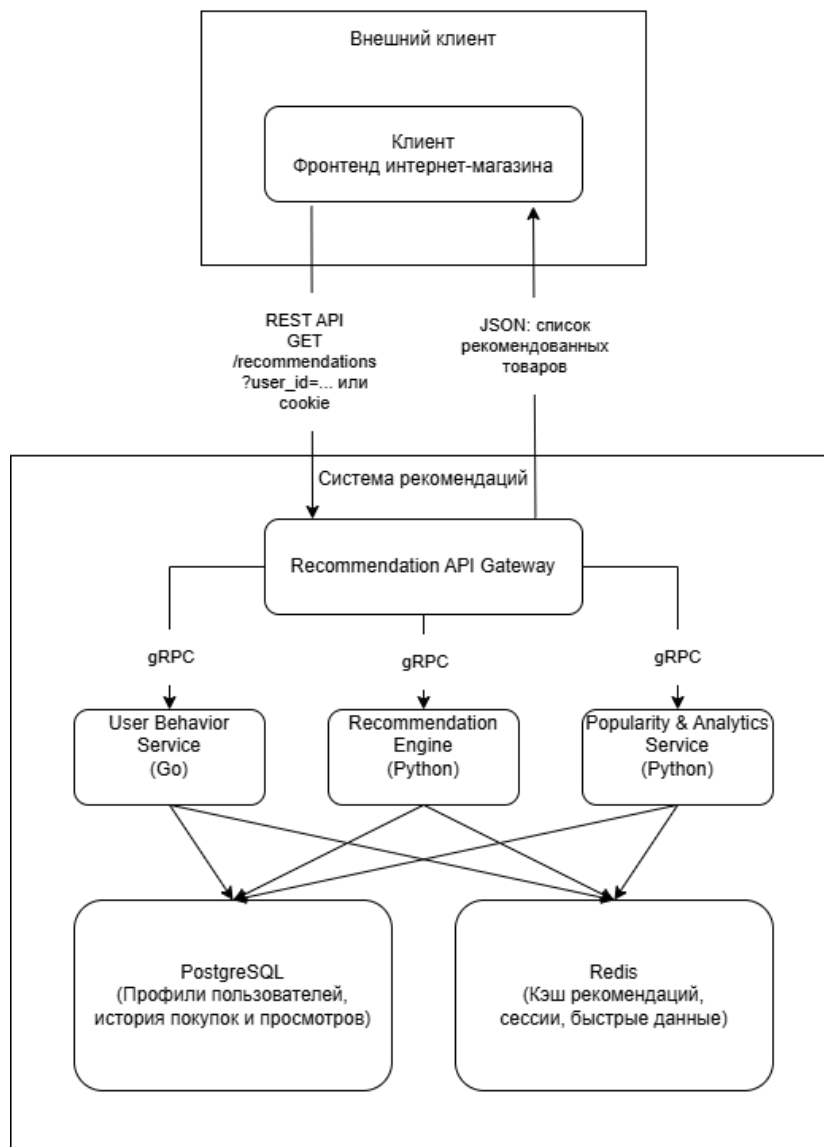


Рисунок 2. Схема взаимодействия микросервисов

3. Методы управления

3.1 Выбор и обоснование методологии

Для реализации проекта адаптивной системы рекомендаций для интернет-магазина выбран гибридный подход Scrum-Kanban, который сочетает структурированность Scrum с гибкостью и визуализацией потока задач Kanban.

Данный выбор обоснован следующими факторами:

1. Итеративная разработка продукта со средней степенью неопределённости. Проект включает компоненты с разной предсказуемостью: разработка Recommendation API Gateway и User Behavior Service на Go относительно линейна и предсказуема, в то время как Recommendation Engine и Popularity & Analytics Service на Python требуют экспериментов с алгоритмами рекомендаций, подбора библиотек и настройки метрик точности. Scrum позволяет работать фиксированными итерациями с демонстрацией рабочего прогресса, а Kanban - ограничивать незавершённую работу и быстро перераспределять усилия.
2. Необходимость оперативного реагирования на изменения. Требования к системе рекомендаций могут уточняться в процессе разработки. Гибкие методологии позволяют легко вносить изменения в бэклог без полной переработки плана проекта.
3. Параллельная работа разнородных задач и специалистов. В проекте задействованы разные технологии (Go и Python) и типы задач (backend-разработка, работа с данными, настройка инфраструктуры Docker/PostgreSQL/Redis). Scrumban позволяет визуализировать поток задач на общей Kanban-доске, синхронизировать усилия через ежедневные стендапы и спринты, а также параллельно вести разработку микросервисов.

Основой подхода является концепция инкремента и спринта.

Инкремент продукта — это завершённая, протестированная и потенциально релизная часть системы, созданная за один спринт. Каждый инкремент должен быть интегрирован с предыдущими, работать в связке с другими сервисами и соответствовать критериям готовности, установленным командой.

Спринт - фиксированный по времени итерационный цикл, в течение которого команда создаёт релизный инкремент продукта и обеспечивает предсказуемость разработки.

Примеры спринтов:

Спринт 1: Анализ требований, проектирование архитектуры, настройка репозитория и базовой инфраструктуры (Docker, Docker Compose). Результат - рабочая среда разработки и утверждённый бэклог.

Спринт 2: Разработка Recommendation API Gateway на Go с базовыми endpoint'ами и обработкой cookie. Результат - работающий шлюз, принимающий запросы и возвращающий заглушенные рекомендации.

Спринт 3: Разработка User Behavior Service и Recommendation Engine на Python с базовой логикой рекомендаций по просмотрам и популярным товарам. Результат - персонализированные рекомендации для авторизованных пользователей на тестовых данных.

Спринт 4: Интеграция всех микросервисов, добавление rule-based правил (учёт цены, дня зарплаты, средних трат), нагрузочное тестирование. Результат - полностью интегрированная система, готовая к демонстрации и внедрению.

4. Организация рабочего пространства управления проектом

4.1 Принципы организации рабочего пространства

Рабочее пространство управления проектом строится на основе принципов гибкости, прозрачности и минимизации рутинных операций, что особенно важно при использовании методологии Scrumban. Ключевые принципы:

1. Единый источник правды: вся информация о задачах, прогрессе, коде и документации хранится в централизованных местах, чтобы избежать дублирования, противоречий и потери данных.
2. Прозрачность и визуализация: каждый член команды в любой момент видит текущее состояние всех задач, их приоритеты и блокеры через общую доску.
3. Самообслуживание: участники команды имеют полный доступ к необходимым инструментам и информации без лишних согласований.
4. Автоматизация рутины: процессы сборки, тестирования и развёртывания автоматизированы, чтобы сосредоточиться на разработке, а не на повторяющихся действиях.

4.2 Инструменты организации рабочего пространства

В качестве центрального инструмента управления задачами и прогрессом проекта выбран GitHub Projects. Эта платформа объединяет управление кодом, задачами и автоматизацию в одном месте, что идеально подходит для небольшого проекта с микросервисной архитектурой.

1. GitHub Repository служит единым источником всего исходного кода проекта.
2. GitHub Projects используется как Kanban-доска для визуализации задач по колонкам: To Do → In Progress → Review → Done. Задачи

привязаны к Issues и Pull Requests, что обеспечивает полную трассируемость от требования до кода.

3. GitHub Actions отвечает за автоматизацию CI/CD: автоматический запуск unit-тестов, линтинга, сборки Docker-образов и локального запуска системы через Docker Compose при каждом Pull Request.
4. GitHub Container Registry используется для хранения собранных Docker-образов микросервисов.

Основной рабочий процесс построен вокруг Pull Request:

1. Любые изменения вносятся через ветку и Pull Request.
2. Перед слиянием обязательно проводится код-ревью и проходят автоматические проверки GitHub Actions.
3. После слияния в main автоматически собираются и пушатся новые Docker-образы.

Docker и Docker Compose обеспечивают воспроизводимость окружения разработки и тестирования: один файл `docker-compose.yml` описывает запуск всех четырёх микросервисов вместе с PostgreSQL и Redis. Это позволяет быстро поднять полную систему локально.

5. Система контроля и мониторинга реализации проекта

5.1 Философия и принципы системы контроля

Система контроля и мониторинга реализации проекта строится на концепции непрерывной обратной связи, раннего обнаружения отклонений и управления на основе данных. Ключевой принцип - измерять только то, что действительно влияет на успех проекта и качество конечного продукта, а не то, что просто легко измерить.

Контроль осуществляется на трёх взаимосвязанных уровнях: стратегическом, тактическом и операционном.

Все метрики и показатели подчинены критериям SMART.

Стратегический контроль отвечает на вопрос «движется ли проект в правильном направлении?». Он оценивает общий прогресс по достижению цели - созданию рабочей адаптивной системы рекомендаций, способной генерировать персонализированные предложения с учётом истории пользователя, популярности товаров, ценового сегмента, дня зарплаты и средних трат.

Тактический контроль сосредоточен на соблюдении методологии Scrumban и сроков. Основные инструменты: состояние Kanban-доски в GitHub Projects, velocity, соблюдение лимитов WIP. Регулярные события позволяют быстро выявлять и устранять блокеры.

Операционный контроль обеспечивает качество и надёжность разрабатываемой системы на уровне кода и инфраструктуры. Ключевые метрики: покрытие кода unit-тестами, успешное прохождение GitHub Actions (CI/CD), время отклика рекомендаций в локальном окружении, отсутствие критических ошибок при запуске через Docker Compose, стабильность подключений к PostgreSQL и Redis.

6. Оценка затрат на реализацию и внедрение

Оценка затрат на реализацию проекта проводится исходя из предположения, что разработка ведётся небольшой командой в течение 4–5 месяцев. Затраты рассчитываются на основе средних рыночных ставок junior/middle-специалистов в России на 2025 год.

6.1 Оценка затрат на реализацию

1. Backend-разработчик (Go): отвечает за Recommendation API Gateway и User Behavior Service.

Участие весь цикл проекта (5 месяцев).

Примерно 140 000 руб./мес.

Итого: $140\,000 \times 5 = 700\,000$ руб.

2. Data/Python-разработчик: отвечает за Recommendation Engine и Popularity & Analytics Service на Python, реализацию алгоритмов рекомендаций.

Участие весь цикл проекта (5 месяцев).

Примерно 160 000 руб./мес.

Итого: $160\,000 \times 5 = 800\,000$ руб.

3. DevOps-инженер: настройка Docker, Docker Compose, PostgreSQL, Redis, CI/CD в GitHub Actions.

Участие частично — 3 месяца.

Примерно 150 000 руб./мес.

Итого: $150\,000 \times 3 = 450\,000$ руб.

4. Project Manager / Product Owner: планирование спринтов, ведение бэклога, документация.

Участие весь цикл проекта (5 месяцев).

Примерно 100 000 руб./мес.

Итого: $100\,000 \times 5 = 500\,000$ руб.

Итого чистые расходы на зарплату: $700\,000 + 800\,000 + 450\,000 + 500\,000 = 2\,450\,000$ руб.

В реальном проекте к этой сумме добавляются налоги, накладные расходы и возможные премии, что может увеличить общую стоимость на 30–50%.

6.2 Оценка затрат на внедрение

Оценка затрат на внедрение и эксплуатацию системы значительно ниже затрат на разработку и сильно зависит от выбранной инфраструктуры.

Основные статьи расходов:

1. Облачные серверы: от 20 000 до 80 000 руб./мес. в зависимости от нагрузки.
2. Базы данных: управляемые PostgreSQL и Redis в облаке 5 000 - 15 000 руб./мес.
3. Мониторинг и логирование 5 000 - 10 000 руб./мес.
4. Домен, SSL, CDN 2 000 - 5 000 руб./мес.
5. Резервное копирование - 3 000 - 10 000 руб./мес.

В первый год эксплуатации суммарные затраты на инфраструктуру составят примерно 300 000–800 000 руб. (в среднем 40 000–60 000 руб./мес.).

7. Анализ рисков и меры по их предотвращению

Реализация проекта адаптивной системы рекомендаций связана с рядом рисков, которые можно разделить на технические и внешние. Для каждого риска оценивается вероятность возникновения и степень влияния на проект. Также предлагаются меры по предотвращению и минимизации последствий.

7.1 Технические риски

1. Недостаточная точность и релевантность рекомендаций. Алгоритм может генерировать нерелевантные предложения, что снизит ценность системы. Высокая вероятность, высокое влияние.

Меры: Использование проверенных библиотек и гибридного подхода. Тестирование на синтетических и реальных датасетах. Внедрение метрик оценки на ранних спринтах и итеративная доработка.

2. Проблемы с производительностью и масштабируемостью. Высокая нагрузка на Recommendation Engine или задержки при запросах к Redis/PostgreSQL могут привести к долгому времени отклика. Средняя вероятность, высокое влияние.

Меры: Раннее профилирование кода, использование кэширования в Redis для часто запрашиваемых рекомендаций. Нагрузочное тестирование начиная с третьего спринта. Оптимизация запросов и асинхронная обработка в Go-сервисах.

3. Сложности интеграции микросервисов и работы с Docker. Проблемы коммуникации между сервисами, ошибки в docker-compose.yml или несовместимость версий зависимостей. Средняя вероятность, среднее влияние.

Меры: Использование единого docker-compose для локальной разработки и тестирования интеграции на каждом спринте. Автоматизация проверок через GitHub Actions.

4. Ошибки обработки cookie и сессий неавторизованных пользователей. Потеря данных о просмотрах из-за неправильной работы с cookie или блокировщиков. Средняя вероятность, среднее влияние.

Меры: Тестирование на разных браузерах, использование fallback-механизмов, ограничение срока жизни cookie в соответствии с политикой конфиденциальности.

7.2 Внешние риски

1. Изменения в API или структуре данных существующего интернет-магазина. Обновление внешнего API товаров или пользовательских профилей может нарушить интеграцию. Низкая вероятность, критическое влияние.

Меры: Выделение отдельного адаптерного слоя в Recommendation API Gateway. Реализация версионирования и автоматических тестов интеграции. Мониторинг изменений внешнего API и подготовка механизма быстрых обновлений.

2. Зависимость от внешних библиотек и сервисов. Обновления библиотек Python или проблемы с Docker Hub/GitHub могут временно блокировать разработку. Средняя вероятность, среднее влияние.

Меры: Фиксация версий зависимостей в requirements.txt и go.mod. Локальное кэширование образов Docker. Резервные варианты библиотек для ключевых функций.

3. Недостаток тестовых данных или проблемы с конфиденциальностью. Отсутствие реальных данных о покупках и зарплатах может затруднить отладку алгоритма. Средняя вероятность, среднее влияние.
- Меры: Использование открытых датасетов и генерация синтетических данных. Соблюдение принципов *privacy-by-design* (хранение только необходимого минимума данных).

Заключение

В процессе выполнения курсовой работы разработан полный план создания адаптивной системы рекомендаций для интернет-магазина. Система генерирует персонализированные предложения товаров с учётом истории просмотров и покупок, популярности, ценового сегмента, дня зарплаты и средних трат, а также на основе cookie.

Проект охватил все этапы конструирования ПО: анализ предметной области, требования, микросервисную архитектуру, декомпозицию, Scrumban-методологию, управление задачами в GitHub, контроль, оценку затрат и анализ рисков.

Список источников и литературы

1. Статья Habr “Интерактивные и документированные диаграммы для сложных систем” [Электронный ресурс] Режим доступа: <https://habr.com/ru/articles/803671/>
2. Статья Habr “Основы Scrum менее чем за 10 минут (Scrum Alliance)” [Электронный ресурс] Режим доступа: <https://habr.com/ru/articles/762828/>
3. Статья Habr “CI/CD на GitHub Actions и GitLab CI для самых маленьких. Часть 1” [Электронный ресурс] Режим доступа: <https://habr.com/ru/articles/914560/>
4. Статья Habr “Инструкция по планированию сложных цифровых проектов” [Электронный ресурс] Режим доступа: <https://habr.com/ru/articles/707264/>
5. Статья Habr “Как проанализировать риски: 4 шага” [Электронный ресурс] Режим доступа: <https://habr.com/ru/companies/otus/articles/806975/>