МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ САМОРАЗВИТИЯ С ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМОЙ АДАПТАЦИИ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Стеклянникова Петра Сергеевича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2026

Актуальность темы. Современные цифровые платформы для саморазвития всё чаще нуждаются в персонализации: пользователи формулируют цели обычным языком, а традиционный поиск по ключевым словам не понимает смысловых связей. Запросы «хочу стать спортивным», «начать бегать» и «улучшить физическую форму» описывают близкую потребность, но для классического текстового поиска это разные запросы. Это определяет потребность в адаптивных системах, объединяющих интеллектуальные рекомендации и автоматическую генерацию контента.

Разработка адаптивных систем для саморазвития представляет собой актуальное направление на стыке программной инженерии, машинного обучения и природу мотивации. Наблюдается устойчивый рост спроса на цифровые платформы, которые помогают пользователям ставить цели, отслеживать прогресс и получать персонализированные рекомендации. Геймификация и системы мотивации показывают высокую эффективность в повышении вовлечённости пользователей. Персонализированные системы демонстрируют значительное преимущество по сравнению с универсальными решениями за счёт адаптации под индивидуальные потребности.

С технологической точки зрения актуальность темы подчёркивается созреванием необходимых предпосылок: широкое распространение облачных вычислений, развитие микросервисных архитектур, прогресс в области обработки естественного языка и доступность предобученных моделей создают уникальную возможность для построения действительно интеллектуальных адаптивных систем.

Цель работы — разработка адаптивного веб-приложения для саморазвития с интеллектуальной системой рекомендаций и генерации персонализированного контента.

Задачи работы:

1. Провести анализ предметной области, современных подходов к построению адаптивных систем и алгоритмов рекомендаций.
2. Спроектировать микросервисную архитектуру, обосновать выбор технологического стека.
3. Выполнить сравнительное тестирование алгоритмов BERT, TF-IDF, Collaborative Filtering и оценить их качество.
4. Реализовать веб-интерфейс, бэкенд-сервис, ML-сервис с алгоритма-

ми семантического поиска и рекомендаций, интегрировать генеративную LLM.

Объект исследования — веб-приложения для саморазвития и постановки целей. **Предмет исследования** — алгоритмы семантического поиска и персонализированных рекомендаций на основе машинного обучения.

Научная новизна работы заключается в сравнительном анализе трёх подходов к рекомендациям (TF-IDF, Collaborative Filtering, BERT) на едином наборе данных с оценкой по четырём метрикам, а также в интеграции семантических рекомендаций с генеративной LLM в рамках одного приложения.

Практическая значимость состоит в создании полнофункционального веб-приложения, готового к использованию. Код проекта открыт и доступен на GitHub.

Анализ предметной области. Collaborative Filtering является одним из наиболее распространённых подходов к построению систем рекомендаций. Основная идея метода заключается в использовании поведения и предпочтений других пользователей для предсказания интересов целевого пользователя. Алгоритм K-ближайших соседей (KNN) находит пользователей или элементы, наиболее похожие на целевой объект, на основе метрик сходства (косинусное сходство, корреляция Пирсона). Преимуществами KNN являются простота реализации и интерпретируемость результатов. Однако метод страдает от проблемы разреженности данных и холодного старта для новых пользователей или элементов.

TF-IDF является классическим методом для оценки важности терминов в документах относительно коллекции документов. Формула определяется как произведение частоты термина в документе на обратную частоту документа в коллекции. TF-IDF широко используется для текстового поиска и ранжирования документов, позволяет находить элементы с похожим текстовым содержанием. Однако метод не учитывает семантическое сходство и синонимию, работает только с точными совпадениями терминов.

BERT представляет собой модель-трансформер, предобученную на больших корпусах текстов. BERT способен понимать контекст слов в обоих направлениях, что позволяет ему лучше улавливать семантические связи по сравнению с традиционными методами. Для задач семантического поис-

ка BERT позволяет находить документы, семантически похожие на запрос, даже при отсутствии точных совпадений терминов. Применение BERT для рекомендаций требует вычисления эмбеддингов текстовых описаний и использования метрик сходства, например, косинусного сходства.

В рамках данной работы был проведён сравнительный анализ различных подходов к рекомендациям. Хотя теоретически гибридные методы могут показывать лучшие результаты, для данной системы было принято решение использовать только BERT. Это обусловлено тем, что BERT демонстрирует достаточную точность, использование единого алгоритма упрощает архитектуру, а также снижает вычислительную нагрузку. BERT способен одновременно учитывать семантику контента и выявлять паттерны в данных пользователей через семантические профили, что делает его универсальным решением для всех задач системы (поиск квестов, рекомендация квестов, рекомендация друзей).

Современные коммерческие системы рекомендаций (Amazon, Netflix, Spotify, YouTube) используют комбинации различных алгоритмов, включая Collaborative Filtering, матричную факторизацию и глубокое обучение. Однако детали реализации таких систем являются проприетарными и не раскрываются, что затрудняет воспроизведение результатов. Открытые реализации, демонстрирующие сравнительный анализ эффективности различных методов в единой системе, встречаются редко, что определяет актуальность данной работы.

Архитектура системы. Разработанное приложение построено на микросервисной архитектуре. Система организована в четыре слоя: Frontend Layer (HTML + JavaScript), Core Backend Layer (Go + Gin + PostgreSQL), ML Service Layer (Python + FastAPI) и LLM Layer (интеграция с Kimi-K2-Thinking).

Основной бэкенд реализован на Go с использованием фреймворка Gin. Выбор Go обусловлен его отличной поддержкой конкурентности, что критически важно для системы, обрабатывающей множество одновременных запросов к сервису рекомендаций. Сервис рекомендаций вынесен в отдельный микросервис на Python с FastAPI. Выбор Python обусловлен богатой экосистемой библиотек для машинного обучения и простотой интеграции предобученных моделей. В процессе разработки также исполь-

зовались scikit-learn для реализации TF-IDF и Collaborative Filtering для сравнительного тестирования, но в итоговой версии используется только BERT.

В качестве основной базы данных выбрана PostgreSQL — надёжная реляционная СУБД с поддержкой сложных запросов и транзакций. Для сервиса рекомендаций используется SQLite: файловая база данных, простая в развёртывании, с возможностью хранения бинарных данных (BLOB) для эмбеддингов. На старте приложения данные загружаются из SQLite в кэш для быстрого доступа и далее алгоритмы рекомендаций и поиска извлекают данные только из кэша. Такая реализация обеспечивает одновременно высокую производительность, консистентность данных и простоту реализации. Отдельный экземпляр PostgreSQL с расширением pgvector или специализированная векторная база данных были бы избыточны для текущего масштаба: при количестве квестов в несколько сотен прямой перебор эмбеддингов в памяти работает быстрее, чем обращение к внешней базе данных по сети.

Фронтенд разработан на HTML, CSS и JavaScript без использования фреймворков. Приложение использует современные библиотеки для специфических задач: FullCalendar для календаря, Cytoscape.js для визуализации графов квестов.

Взаимодействие между бэкендом и сервисом рекомендаций осуществляется через REST API с таймаутом 30 секунд. Некритичные операции, такие как добавление квестов в индекс рекомендаций, выполняются асинхронно через горутины. Такое разделение позволяет независимо масштабировать компоненты и обновлять ML-модель без остановки основного сервиса.

Архитектура базы данных. Спроектирована реляционная база данных на PostgreSQL из 11 таблиц, организованных в логические группы. В группу «Пользователи и прогресс» входят таблица users с учётными данными и статистикой (xp_points, coin_balance, level) и таблица user_coin_transactions для аудита финансовых операций. Группа «Задачи и выполнение» включает таблицу tasks (каталог задач с метаданными и наградами), таблицу user_tasks (связь пользователей с задачами, статусы выполнения, расписание) и таблицу quest_tasks (композиция квестов из задач с указанием

порядка). Квестовая система представлена таблицами quests (метаданные квестов, условия прохождения, награды, бонусы в JSONB) и user_quests (прогресс пользователей, статусы, временные метки). Социальные взаимодействия реализованы через таблицы friends (двунаправленные связи) и shared_quests (совместные квесты с синхронным завершением).

Слоистая архитектура бэкенда. Бэкенд-приложение организовано по принципам чистой архитектуры и разделено на четыре слоя. Handler Layer обрабатывает HTTP-запросы, валидирует входные данные через Gin binding, извлекает параметры и вызывает методы сервисного слоя. Service Layer содержит бизнес-логику, координирует работу репозиториев и интегрируется с внешними системами. Repository Layer абстрагирует доступ к базе данных, выполняет SQL-запросы, управляет транзакциями и маппит данные в структуры Go. Model Layer определяет структуры данных, валидирует бизнес-правила и обеспечивает сериализацию JSON. Такое разделение обеспечивает тестируемость, поддерживаемость, переиспользуемость и масштабируемость.

Основной бэкенд. Бэкенд покрывает весь жизненный цикл пользователя. Реализована система аутентификации и авторизации на основе JWT с безопасным хранением паролей через bcrypt с параметром стоимости 10. При успешной аутентификации генерируется токен с временем жизни 45 минут. Токен содержит идентификатор пользователя в payload и подписывается секретным ключом. Middleware JWTAuthMiddleware проверяет наличие и валидность токена в заголовке Authorization каждого защищённого запроса.

Реализована система уровней с квадратичной прогрессией: $level = \lfloor \sqrt{XP/100} \rfloor + 1$. Формула обеспечивает экспоненциальный рост требований к опыту для каждого следующего уровня: уровень 1 — 0–99 XP, уровень 2 — 100–399 XP, уровень 3 — 400–899 XP, уровень 4 — 900–1599 XP и так далее. При выполнении задачи или завершении квеста опыт начисляется пользователю, и уровень автоматически пересчитывается. Обновление уровня происходит атомарно в рамках транзакции вместе с начислением опыта и монет.

Квесты фильтруются по уровню сложности для обеспечения постепенного открытия контента. Алгоритм фильтрации реализован в методе

GetAvailableQuests: пользователь видит только те квесты, сложность которых не более чем на один уровень выше его текущего уровня, цена которых не превышает его баланс и которые ещё не были куплены или пройдены.

Все критические операции выполняются в ACID-транзакциях PostgreSQL с использованием метода BeginTxx для создания транзакций с контекстом. Процесс покупки квеста: начало транзакции, проверка существования квеста и достаточности средств, создание записи в user_quests со статусом purchased, создание записей в user_tasks для всех задач квеста со статусом not_started, списание монет, создание записи в user_coin_transactions для аудита, подтверждение транзакции. Процесс выполнения задачи включает автоматический старт квеста, получение базовых наград, начисление наград с автоматическим пересчётом уровня и обновление статусов.

Реализована система социальных взаимодействий: добавление друзей через указание username и создание совместных квестов с синхронным завершением. Квест считается завершённым только когда все задачи выполнены обоими пользователями. При завершении квеста одним пользователем система проверяет статус у второго. Все операции выполняются атомарно в рамках одной транзакции.

Финансовые операции логируются в таблице user_coin_transactions, которая содержит идентификатор пользователя, сумму, тип транзакции, тип связанной сущности, её идентификатор, текстовое описание и временную метку.

Интеграция с LLM. Реализована интеграция с LLM API (Intelligence.IO Solutions, модель Kimi-K2-Thinking). Пользователь отправляет текстовый промпт с описанием желаемого квеста через эндпоинт POST /quests/generate. Создаётся детальный системный промпт, определяющий структуру JSON-ответа: метаданные квеста (название, описание, категория, редкость, сложность, цена, награды) и массив задач с параметрами. Запрос отправляется к API с параметрами model и temperature 0.7. Ответ очищается от thinking-тегов, парсится в структуру AIQuestResponse и сохраняется в базу данных. Сгенерированный контент автоматически добавляется в индекс рекомендаций через асинхронный запрос в горутине. Модель обладает встроенной системой цензуры, блокирующей генерацию вредного контента.

Дополнительно реализована функция генерации календаря задач: система собирает информацию об активных квестах пользователя и через LLM генерирует оптимальное расписание с учётом дедлайнов и последовательности выполнения. Сгенерированное расписание применяется через INSERT ... ON CONFLICT DO UPDATE для атомарного обновления.

Сервис рекомендаций. Сервис реализован на Python с FastAPI с использованием архитектуры, основанной на состоянии приложения (app.state). При инициализации загружается предобученная модель `paraphrase-multilingual-MiniLM-L12-v2` из библиотеки sentence-transformers. Модель автоматически использует GPU, если доступен, иначе CPU. Создаются кэшируемые структуры данных: app.state.quest_embeddings (словарь эмбеддингов квестов), app.state.quests_data (метаданные), app.state.users_data (данные пользователей), app.state.profile_embeddings (семантические профили). При старте все данные загружаются из SQLite в кэш в памяти.

Для каждого квеста формируется текстовое представление путём объединения названия, описания и категории:

$$\mathbf{e}_d = \text{BERT}(\text{concat}(\text{title}_d, \text{description}_d, \text{category}_d))$$

Семантический поиск: запрос пользователя векторизуется в $\mathbf{e}_q$, после чего для всех квестов вычисляется косинусное сходство $\cos(\mathbf{e}_q, \mathbf{e}_d)$. Результаты ранжируются по убыванию сходства.

Персонализированные рекомендации: строится семантический профиль пользователя как усреднённый эмбеддинг всех приобретённых им квестов $\mathbf{p}_u = \frac{1}{|U|} \sum_{q \in U} \mathbf{e}_q$. Новые квесты ранжируются по косинусной близости к профилю, уже приобретённые исключаются.

Рекомендация друзей: вычисляется косинусное сходство семантических профилей пользователей $\cos(\mathbf{p}_u, \mathbf{p}_v)$, существующие друзья исключаются из списка. Для каждой рекомендации генерируется объяснение на основе общих категорий квестов.

Все операции поиска и рекомендаций выполняются исключительно в кэше, что устраняет проблему синхронного доступа к SQLite и не блокирует асинхронность FastAPI. Запись новых эмбеддингов происходит асинхронно.

Интеграция с бэкендом. Реализованы следующие HTTP-эндпоинты:

POST /api/quests/add (добавление квестов в индекс), POST /api/users/add (добавление пользователей и построение профилей), POST /api/sync (синхронизация с PostgreSQL), POST /api/search (семантический поиск), POST /api/quests/recommend (рекомендация квестов), POST /api/users/recommend (рекомендация друзей). Все запросы выполняются с таймаутом 30 секунд.

Сравнительный анализ алгоритмов. В рамках работы были протестированы три подхода: TF-IDF, Collaborative Filtering на KNN и BERT. Для оценки использовались метрики Precision@K, Recall@K, F1-score@K и NDCG@K. Для рекомендации квестов используется K=5 (стандартная практика: пользователь просматривает не более пяти результатов). Для рекомендации друзей — K=3 (более строгий критерий для ограниченного списка).

TF-IDF выступил в качестве базового уровня. Векторизация квестов выполнялась с помощью TfidfVectorizer из scikit-learn по названию, описанию и категории с параметром max_features=1000. Сходство между запросом и квестами вычислялось через косинусную меру, результаты ранжировались по убыванию.

Collaborative Filtering использовал бинарную матрицу $M_{n\times m}$ «пользователь–квест», где $M_{i,j} = 1$, если пользователь приобрёл квест. Для целевого пользователя находились K ближайших соседей по косинусному сходству векторов взаимодействий через NearestNeighbors из scikit-learn. Рекомендации формировались из квестов соседей.

BERT использовал семантический подход: описания квестов преобразовывались в эмбеддинги, и близость вычислялась через косинусное сходство. Модель не требует дополнительного обучения.

Результаты тестирования для рекомендации квестов:

Алгоритм	**Precision@5**	**Recall@5**	**F1@5**	**NDCG@5**
TF-IDF	0.480	0.306	0.370	0.482
Collaborative Filtering	0.400	0.246	0.303	0.464
BERT	0.540	0.338	0.412	0.569

BERT продемонстрировал лучшие результаты. Наибольший прирост — по NDCG@5 (+18% относительно TF-IDF), что говорит о более качественном ранжировании. Collaborative Filtering показал наихудшие результаты из-за разреженности матрицы взаимодействий при небольшом

количестве пользователей.

Для рекомендации друзей BERT также показал лучший результат: F1@3 = 0.641 против 0.636 у TF-IDF и 0.631 у Collaborative Filtering. Прирост обеспечен способностью BERT находить семантически близких пользователей даже при отсутствии общих категорий квестов.

Для оценки абсолютного качества метрик они были сопоставлены с результатами, опубликованными в Scientific Reports (Nature, 2024). Значения NDCG@5 для baseline-моделей в этом исследовании находятся в диапазоне 0.45–0.59. Полученный NDCG@5 = 0.569 попадает в этот диапазон, что подтверждает качество разработанной системы.

Тестирование и отказоустойчивость. Для обеспечения качества системы проведено тестирование всех компонентов. Большинство тестов выполнялось вручную, также написаны unit-тесты API сервиса рекомендаций.

Проведено сравнительное тестирование алгоритмов на синтетических данных с вычислением метрик в директории benchmark_comparison_algorithms. Для автоматизированной проверки API разработан скрипт test_api.py, покрывающий все эндпоинты.

Проведены тесты встроенной системы цензуры модели Kimi-K2-Thinking на различных типах нежелательного контента, подтвердившие эффективность фильтрации.

Проведено интеграционное тестирование взаимодействия бэкенда с сервисом рекомендаций, валидация транзакций при различных сценариях, end-to-end тестирование пользовательских сценариев.

Реализована многоуровневая обработка ошибок: на уровне репозитория транзакции откатываются, на уровне сервиса бизнес-ошибки преобразуются в сообщения, на уровне HTTP возвращаются соответствующие статус-коды. Используется структурное логирование через библиотеку slog. Реализованы health check-эндпоинты для PostgreSQL и сервиса рекомендаций.

Управление конфигурацией. Система использует переменные окружения: JWT_SECRET для подписи токенов, API_KEY_INTELLIGENCE_IO для доступа к LLM, DATABASE_URL для подключения к PostgreSQL, RECOMMENDATION_SERVICE_BASE_URL для связи с сервисом рекомендаций. Это обеспечивает безопасность, гибкость для разных окружений

и простоту развёртывания.

Производительность и оптимизация. В сервисе рекомендаций используется кэширование BERT-эмбеддингов в памяти для избежания повторных вычислений. Операции с эмбеддингами выполняются через numpy с векторизацией для batch-обработки. В бэкенде используется SELECT только необходимых полей, батчинг при массовых вставках. Асинхронная обработка некритичных операций через горутины и таймауты для внешних запросов. Микросервисная архитектура и stateless-свойство основного Go-бэкенда обеспечивают горизонтальное масштабирование.

Фронтенд. Фронтенд написан на HTML5, CSS3 и JavaScript без фреймворков, имеет модульную архитектуру. Реализовано семь модулей: аутентификация (управление JWT через localStorage), работа с квестами, семантический поиск с интеграцией сервиса рекомендаций, AI-генерация, друзья, календарь (FullCalendar), карта квестов (Cytoscape.js). Взаимодействие с бэкендом через единую функцию apiCall с автоматической подстановкой токена и обработкой ошибок. Реализована обработка голосового ввода через Web Speech API, адаптивный дизайн с CSS Grid и Flexbox.

Направления дальнейшего развития. Разработанная система может быть расширена в нескольких направлениях. В области алгоритмов — внедрение гибридных методов, fine-tuning BERT на данных системы, применение обучения с подкреплением. В области производительности — использование векторных баз данных (FAISS, Qdrant, Milvus) для поиска похожих эмбеддингов. В области AI — добавление генерации изображений для квестов. В области масштабирования — использование брокеров сообщений (RabbitMQ, Kafka, NATS). В области мониторинга — интеграция Prometheus и Grafana, A/B-тестирование алгоритмов.

Основные результаты работы:

1. Спроектирована микросервисная архитектура веб-приложения с разделением на Go-бэкенд и Python-сервис рекомендаций. Разработана схема базы данных PostgreSQL из 11 таблиц.
2. Реализован основной бэкенд с JWT-аутентификацией, системой уровней с квадратичной прогрессией, транзакционной обработкой критических операций, аудитом финансовых операций, социальными взаимодействиями и слоистой архитектурой.

3. Разработан Сервис Рекомендаций с семантическим поиском и персонализированными рекомендациями на основе BERT, кэшированием эмбеддингов в памяти и двухуровневым хранением данных.
4. Проведён сравнительный анализ TF-IDF, Collaborative Filtering и BERT по четырём метрикам. BERT показал лучшие результаты и выбран основным алгоритмом.
5. Реализована интеграция с LLM для генерации персонализированных квестов и календаря задач.
6. Разработан фронтенд с модульной архитектурой и поддержкой голосового ввода.
7. Проведено тестирование всех компонентов системы, обеспечена отказоустойчивость через таймауты, асинхронную обработку и многоуровневую обработку ошибок.