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

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ ДЛЯ ОБУЧАЮЩЕЙ СИСТЕМЫ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Кириленко Константина Вадимовича

Научный руководитель
зав. каф. техн. пр.,
к. ф.-м. н., доцент _____ С. В. Миронов

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

Саратов 2025

СОДЕРЖАНИЕ

1 Задача разработки web-приложения для обучающей системы

1.1 Постановка задачи

Разработка веб-приложения для изучения языков и терминов направлена на создание универсальной образовательной платформы, сочетающей мультимедийные технологии с адаптивными методиками обучения. Сервис ориентирован на преодоление ограничений традиционных подходов к запоминанию лексики, где превалирующая опора на текстовые форматы снижает эффективность усвоения специализированной терминологии. Цель проекта заключается в интеграции мультисенсорного взаимодействия с учебным материалом, что предполагает синтез визуальных, аудиальных и контекстных элементов в рамках единой обучающей среды.

Ключевой функционал системы охватывает возможность создания тематических курсов с поддержкой разноформатных данных, включая графические иллюстрации, аудиозаписи произношения и видеофрагменты с примерами использования терминов. Особенностью сервиса является автоматическая генерация интерактивных упражнений, основанных на ассоциативном сопоставлении контента: пользователи соотносят слова с изображениями, определяют корректные аудиоинтерпретации фраз или анализируют видеосценарии для закрепления контекстного понимания. В отличие от массовых аналогов, фокусирующихся на общеязыковой подготовке, предлагаемое решение предоставляет инструменты для построения иерархических связей между понятиями внутри конкретной предметной области, что способствует формированию системного понимания дисциплины.

Технологические аспекты разработки предполагают решение задач кроссплатформенной совместимости, оптимизации хранения мультимедийных ресурсов и обеспечения низколатентного взаимодействия с серверными компонентами. Бизнес-модель платформы строится на сочетании бесплатного доступа к базовым курсам с подпиской на расширенные функции, такими как персонализированная аналитика прогресса или доступ к экспертной проверке упражнений.

1.2 Выбор веб-технологий

Выбор в пользу веб-приложения обусловлен комплексом технико-экономических факторов. Статистика распространения веб-стандартов подтверждает их поддержку на 98% пользовательских устройств, что обеспечивает

максимальный охват целевой аудитории. Экономическая эффективность достигается за счёт единой кодовой базы, сокращающей затраты на разработку и сопровождение на 30-40% по сравнению с нативными мобильными решениями [**?**]. Такая архитектура обеспечивает баланс между богатством функциональности и доступностью, соответствующий современным тенденциям в образовательных технологиях.

1.3 Обоснование выбора MVC-архитектуры

Архитектура MVC структурирует приложение через разделение ответственности между тремя ключевыми компонентами. Модель инкапсулирует бизнес-логику и управление данными, включая взаимодействие с хранилищами информации и валидацию учебного контента. Представление отвечает за визуализацию интерфейса обучения, трансформируя данные модели в HTML-шаблоны с поддержкой мультимедийных элементов. Контроллер выступает посредником, обрабатывая пользовательские запросы от упражнений и тестов, синхронизируя изменения между моделью и представлением. Сильной стороной подхода является чёткая декомпозиция, позволяющая параллельно разрабатывать компоненты разными командами. Например, фронтенд-разработчики могут сосредоточиться на оптимизации отображения интерактивных карточек для слов, в то время как бэкенд-специалисты реализуют алгоритмы прогрессивного повторения терминов. Тестирование упрощается за счёт изоляции слоёв — юнит-тесты покрывают модель, а интеграционные проверяют корректность взаимодействия контроллера с внешними сервисами.

Кеш-слой, внедрённый между сервисами и репозиториями, оптимизирует производительность за счёт сокращения повторных запросов к базе данных для часто используемых объектов — метаданных курсов, статистики пользователей и шаблонов упражнений.

Архитектура обеспечивает модульность разработки: изменения в логике хранения данных не затрагивают сервисный слой, а модификация бизнес-правил не требует переписывания контроллеров. Это особенно критично для систем с динамически обновляемым функционалом, где компоненты управления контентом и тестирования развиваются независимо.

Масштабируемость решения поддерживается возможностью выделения ресурсоёмких операций (обработка видео, генерация тестов) в отдельные сервисы без изменения базовой структуры. Использование MVC в качестве ядра системы

создаёт баланс между гибкостью распределённых вычислений и простотой поддержки монолитной архитектуры на начальных этапах развития проекта.

1.4 Выбор технологического стека

Разработка платформы для изучения языков требует выбора инструментов, обеспечивающих производительность, безопасность и масштабируемость. Решения основаны на анализе современных трендов веб-разработки, требованиях к обработке мультимедийного контента и сложных сценариях взаимодействия.

Java выбран благодаря техническим характеристикам и экосистеме. Статическая типизация обеспечивает раннее выявление ошибок, критичное для обработки персональных данных. Механизм проверки исключений структурирует обработку ошибок при работе с внешними хранилищами. Кроссплатформенность JVM позволяет развёртывать приложение на различных инфраструктурах (AWS, Google Cloud), что соответствует требованию масштабируемости. Поддержка многопоточности обеспечивает эффективное распараллеливание задач генерации упражнений и обработки медиаконтента.

Производительность современных JVM (HotSpot, GraalVM) сопоставима с нативными языками благодаря JIT-компиляции. Настраиваемые алгоритмы сборки мусора (G1, ZGC) обеспечивают баланс пропускной способности и времени пауз, критичный для интерактивного тестирования. Безопасность обеспечивается встроенными криптографическими API для шифрования чувствительной информации. Долговременная поддержка (LTS) версий гарантирует стабильность и безопасность.

Spring Boot унифицирует разработку распределённых систем через интегрированную экосистему. Принцип конвенции над конфигурацией минимизирует ручную настройку инфраструктуры. Модульная структура стартеров позволяет избирательно подключать функциональные блоки, сохраняя минимальный объём зависимостей. Архитектура обеспечивает согласованное взаимодействие слоёв приложения. Безопасность реализуется через централизованную систему аутентификации с поддержкой современных стандартов токенизации.

Интеграция с внешними сервисами осуществляется через стандартизированные клиенты. Поддержка распределённого кеширования повышает производительность при частом доступе к метаданным. Фреймворк предоставляет инструменты для комплексного тестирования и поддерживает эволюцию архитектуры от монолита до микросервисов.

PostgreSQL выбран благодаря сочетанию реляционной строгости и гибкости. Поддержка расширенных типов (JSONB) позволяет эффективно моделировать иерархические связи терминов и динамические метаданные курсов. Встроенные механизмы полнотекстового поиска и пространственных индексов обеспечивают высокую производительность при работе с крупными терминологическими базами.

Интеграция Elasticsearch обеспечивает расширенный семантический поиск, превосходящий возможности реляционных СУБД. Анализаторы Elasticsearch обрабатывают морфологию языка, синонимию и устойчивость к опечаткам, обеспечивая точное сопоставление запросов с учебным контентом. Распределённая архитектура позволяет горизонтально масштабировать поисковый кластер, гарантируя отказоустойчивость через репликацию шардов. Нечёткий поиск реализуется комбинацией fuzzy-запросов и n-gram-анализа.

Синхронизация с PostgreSQL осуществляется через Logstash, отслеживающий изменения через JDBC-плагин. Трансформация данных включает денормализацию сущностей и преобразование JSONB-полей. Оффлоад поисковых операций снижает нагрузку на основную БД.

Redis используется как in-memory кеш-слой для снижения нагрузки на БД. Архитектура обеспечивает время отклика <1 мс для операций чтения. Поддержка структур данных (ключ-значение, хэши) оптимизирует хранение метаинформации курсов и сессий пользователей. Автоматическое удаление данных (TTL) гарантирует актуальность контента.

Интеграция с Spring Boot через аннотации `@Cacheable` упрощает реализацию кеширования ресурсоёмких операций. Pub/Sub-механизм синхронизирует кеш между экземплярами приложения. S3-протокол выбран для хранения мультимедиа из-за надёжности и масштабируемости. Объектная модель обеспечивает единообразный доступ к файлам через REST API. Версионирование объектов сохраняет исторические редакции материалов.

Использование MinIO для приватного облака обеспечивает контроль данных. Потоковая загрузка файлов оптимизирует сетевые ресурсы, а предварительные подписи URL дают безопасный временный доступ. Метаданные файлов сохраняются в PostgreSQL для сложных запросов без обращения к хранилищу.

Применение Docker в проекте обусловлено необходимостью обеспечения воспроизводимости окружения и упрощения процесса развёртывания распреде-

лённых компонентов системы. Контейнеризация позволяет изолировать сервисы платформы (бэкенд, базы данных, кеш-хранилища) в независимые среды с гарантированной версионностью зависимостей. Механизм слоёв образов оптимизирует доставку обновлений — модификации затрагивают только изменённые части контейнеров, что сокращает время сборки и объём передаваемых данных.

Использование OpenAPI стандартизирует описание REST API. Автогенерация документации через аннотации синхронизирует документацию с кодом. Swagger UI предоставляет визуализацию схем, тестирование endpoints и валидацию параметров. Интеграция с Spring Boot через `springdoc-openapi` обеспечивает динамическое обновление документации.

React обеспечивает компонентный подход, оптимизируя рендеринг динамических интерфейсов. TypeScript предотвращает ошибки при работе со сложными структурами данных. Связка TanStack Query и Axios реализует фоновое обновление данных, кеширование и централизованную обработку ошибок. Хуки useQuery/useMutation инкапсулируют логику взаимодействия с сервером. Архитектура оптимизирована для мультимедийного контента через ленивую загрузку и предварительную выборку данных.

Ant Design предоставляет комплекс доступных компонентов, соответствующих стандартам юзабилити. Архитектура на основе атомарного дизайна позволяет создавать сложные образовательные сценарии. Интеграция с React и TypeScript обеспечивает строгую типизацию свойств. Механизм темизации адаптирует интерфейс под педагогические задачи. Библиотека поддерживает локализацию, доступность и RTL-языки.

2 Реализация web-приложения для обучающий системы

Контейнеризация приложения реализована средствами Docker 24.0+ с оркестрацией Docker Compose v2.23+. Бэкенд на Java 17 собирается через Gradle 8.13 с многоступенчатой сборкой: этап компиляции на базе образа gradle:8.13-jdk17, финальный образ — на eclipse-temurin:17-jre-jammy. Фронтенд на Node.js 18 использует образ node:18-alpine с сервингом через Nginx Alpine. База данных PostgreSQL 16 развернута на postgres:16-alpine, объектное хранилище — на актуальной версии MinIO. Поисковый кластер ELK 8.17.1 включает Elasticsearch, Kibana и Logstash. Кэширование обеспечивается Redis (redis:alpine), обратное проксирование — Nginx Alpine. Инфраструктура тестировалась на Ubuntu 24.04 с конфигурацией 2×3.3 ГГц CPU, 4 ГБ RAM.

База данных включает семь ключевых таблиц. Таблица `users` хранит учетные данные, включая `email` (уникальный индекс), хеш пароля и статус активации. Ролевая модель управляется через `user_roles`, связывающую пользователей с перечислением `Authorities`. Учебные курсы представлены в `courses` со ссылкой на автора, названием и описанием. Иерархия строится через `levels` с порядковыми номерами. Термины хранятся в `words` с поддержкой аудио/видео вложений и флагом активности. Прогресс обучения отслеживается в `progress` с составным ключом (`user_id`, `word_id`) и процентом усвоения (0–100). Активация аккаунтов реализована через `mail_token` с TTL-очисткой, отзыв JWT-токенов — в `deactivated_token`. Участие в курсах фиксируется в `course_participants`. Каскадное удаление данных обеспечено через `ON DELETE CASCADE`, индексы оптимизируют частые запросы.

REST API реализует CRUD-операции для сущностей через стандартные HTTP-методы. `CourseController` поддерживает создание курсов (`POST`), обновление метаданных (`PUT`), постраничное получение (`GET`) и удаление (`DELETE`). Для слов с мультимедиа используется комбинированный подход с `multipart/form-data`. `ProgressController` предоставляет аналитику успеваемости по словам, уровням и курсам с расчетом на основе весовых коэффициентов. `LearningController` формирует учебные материалы адаптивно, увеличивая частоту показа слабо усвоенных терминов, и проверяет ответы с обновлением прогресса. Аутентификация обрабатывается фильтрами `Spring Security`, эндпоинты `/api/login` и `/api/logout` документируются в `Swagger`. Ролевая модель реализована через `@PreAuthorize` на уровне сервисов.

Система аутентификации использует JWT с разделением access/refresh-токенов. Access-токены (срок жизни 15 минут) содержат идентификатор пользователя и роли, подписываются `HMAC-SHA256`. Refresh-токены (срок 7 дней) шифруются алгоритмом `Direct Encryption`. Генерация токенов выполняется фабриками с использованием `MACSigner` для access-токенов и `DirectEncrypter` для refresh. Обработка запросов осуществляется каскадом фильтров: `JwtAuthenticationFilter` извлекает и валидирует токены, `RequestJwtTokensFilter` обрабатывает вход, `RefreshTokenFilter` обновляет access-токены, `JwtLogoutFilter` инвалидирует токены при выходе. Для паролей применяется `BCryptPasswordEncoder`. При успешной аутентификации создается связанная пара токенов, где access-токен сохраняет идентификатор исходного refresh-токена.

Сервисы реализуют CRUD для курсов, уровней и слов. `createCourse` связывает курс с автором через аутентификацию, `updateCourse` проверяет права через `checkAuthorOrAdmin`. Кеширование через `@Cacheable`/`@CacheEvict` оптимизирует доступ к данным:

```
@Caching(evict = {
  @CacheEvict(value="levelDetails", key="#levelId"),
  @CacheEvict(value="courseLevels", key="#level.course.id")
})
```

ElasticSearch обеспечивает полнотекстовый поиск. Управление структурой включает автоматический расчёт порядка уровней (`calculateNextOrderNumber`) и каскадное удаление. Медиафайлы терминов загружаются в MinIO через `S3Service`:

```
private String processFileUpload(MultipartFile file) {
  return s3Service.uploadFile(file);
}
```

Алгоритм адаптивного обучения (`selectWordBasedOnProgress`) использует взвешенный выбор: слова с низким прогрессом (0-100%) получают вес `100 - текущий_прогресс`. Варианты ответов формируются по типу задания (`WORD_TO_IMAGE`/`IMAGE_TO_WORD`), перемешиваются. Проверка ответов обновляет прогресс: ±10% с ограничением `Math.min`/`Math.max`. Средние показатели рассчитываются агрегационными запросами:

```
private Double getAverageProgressForLevel(Long userId, Long levelId) {
  return (double) progressRepository.getTotalProgress() / wordRepository.count();
}
```

Регистрация генерирует 6-значный код (криптографически безопасный), сохраняет с TTL, деактивирует предыдущие коды. ‘EmailService‘ отправляет письмо асинхронно (Thymeleaf-шаблоны). Активация через проверку кода, срока и статуса. Неактивированные пользователи блокируются в ‘UserDetailsService‘ (поле ‘isEnabled‘).

CRUD курсов: `createCourse` связывает с автором, `updateCourse` проверяет права через `checkAuthorOrAdmin`. Кеширование: `@Cacheable("allCourses")`, `@CacheEvict`. Поиск через `ElasticSearch`:

```
@Cacheable(value = "courses", key = "#query")
public List<CourseResponseDTO> getCourses(String query) {
    return courseDocumentRepository.searchCourses(query)
        .stream()
        .map(courseMapper::toDto)
        .collect(Collectors.toList());
}
```

Управление структурой: `calculateNextOrderNumber` определяет позицию уровня. При удалении (`deleteLevel`) пересчитываются порядковые номера. Каскадное кеширование:

```
@Caching(evict = {
    @CacheEvict(value = "levelDetails", key = "#levelId"),
    @CacheEvict(value = "courseLevels", key = "#level.course.id"),
    @CacheEvict(value = "words", allEntries = true)
})
public void deleteLevel(Long levelId, UserDetails userDetails) {
    // Логика удаления
}
```

Медиафайлы загружаются в S3 через `S3Service`:

```
private String processFileUpload(MultipartFile file) {
    return Optional.ofNullable(file)
        .filter(f -> !f.isEmpty())
        .map(s3Service::uploadFile)
        .orElse(null);
}
```

Адаптивный подбор заданий: взвешенный выбор (`selectWordBasedOnProgress`). Вес = 100 - текущий прогресс:

```
private Word selectWordBasedOnProgress(List<Word> words, Long userId) {
    Map<Word, Integer> weights = new HashMap<>();
    for (Word word : words) {
```

```
        int progress = ... // Запрос прогресса
        weights.put(word, 100 - progress);
    }
    return WeightedRandomSelector.select(words, weights);
}
```

Прогресс кешируется:

```
@Cacheable(value = "userWordProgress", key = "{#userId, #wordId}")
public Integer getProgress(Long userId, Long wordId) {
    return progressRepository.findByUserIdAndWordId(...)
        .map(Progress::getProgressValue)
        .orElse(0);
}
```

2.1 Работа с данными и интеграция хранилищ

Модуль `S3Service` обеспечивает загрузку мультимедийных ресурсов в объектное хранилище MinIO. Генерация уникального ключа файла сочетает UUID и оригинальное имя, обеспечивая уникальность и предотвращая коллизии. Автоматическое определение MIME-типа гарантирует корректную обработку контента, а формирование публичных URL обеспечивает прямой доступ к ресурсам. Обработка ошибок ввода-вывода реализована через кастомное исключение `ApiException`.

Интеграция с Elasticsearch осуществляется через Java API Client с поддержкой сложных запросов. Поисковая система использует взвешенные условия с коэффициентами значимости, fuzzy-логику для автоматического исправления опечаток и ngram-анализаторы для частичных совпадений. Nested-запросы позволяют работать с иерархическими структурами данных (курс → уровни → слова), сохраняя контекстную связность.

```
// Взвешенный поиск с fuzzy-логикой
BoolQuery.Builder boolQuery = new BoolQuery.Builder()
    .should(sh -> sh.match(m ->
    ↪  m.field("title.ngram").query(query).fuzziness("AUTO").boost(3.0f));
```

Доступ к данным реализован через Spring Data JPA репозитории. Автоматическая генерация запросов на основе соглашений об именовании (например, `findByCourseIdOrderByOrderNumberAsc`) сочетается с кастомными JPQL-запросами для сложной бизнес-логики. Гибридный подход позволяет эффективно работать с агрегационными функциями и вложенными сущностями.

```
// Кастомный запрос для проверки токенов
@Query("SELECT CASE WHEN COUNT(d) > 0 THEN true ELSE false END " +
       "FROM DeactivatedToken d WHERE d.id = :tokenId AND d.keepUntil > CURRENT_TIMESTAMP")
boolean existsActiveDeactivatedToken(@Param("tokenId") UUID tokenId);
```

Синхронизация между PostgreSQL и Elasticsearch настроена через Logstash. Ежеминутный JDBC-поллинг обеспечивает актуальность данных, а преобразование JSON-структур сохраняет иерархические связи. Шаблон маппинга в Elasticsearch определяет строгую схему данных с мультиязычными анализаторами и nested-типами для сложных объектов.

2.2 Развертывание сервиса

Архитектура развертывания основана на Docker Compose, объединяющем шесть сервисов: PostgreSQL, Elasticsearch, Spring Boot приложение, React фронтенд, Logstash и Nginx. Контейнеризация обеспечивает изоляцию компонентов через bridge-сеть `app-network`, оптимизируя взаимодействие между слоями приложения.

Обратный прокси Nginx обеспечивает SSL-шифрование через Let's Encrypt и балансировку нагрузки. Конфигурация безопасности включает ограничение доступа к Elasticsearch внутренним DNS-именем `elasticsearch:9200`, блокируя внешние подключения. Политика перезапуска `restart: unless-stopped` минимизирует простои, обеспечивая высокую доступность сервиса.

Интеграция Logstash настроена как зависимый сервис, гарантирующий последовательную инициализацию компонентов. Настройка `bootstrap.memory_lock: true` для Elasticsearch предотвращает свопинг памяти, повышая производительность поисковых операций. Сетевая изоляция и разделение обязанностей между контейнерами соответствуют принципам микросервисной архитектуры.