ТОО «Колледж Хекслет»

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

на тему: «Разработка веб-приложения "КнигаРецептов" с использованием облачной базы данных»

по модулю: «Разработка модулей программного обеспечения

для компьютерных систем»

Специальность: 06130100 – Программное обеспечение (по видам)

Квалификация: 4S06130105 – Техник информационных систем

Выполнил(-а): студент 2 курса

7ТИС группы

Ескали Бейбарыс Женисулы

Допущен(-а) к защите «___» _____ 20___ г.

Преподаватель: _____

Защитил(-а) с оценкой: _____ / _____ / _____

(буква) / (циф.экв.) / (балл)

Алматы 2025

ВВЕДЕНИЕ

Современный мир характеризуется стремительным развитием веб-технологий и облачных вычислений. Кулинарные ресурсы занимают одно из ведущих мест среди наиболее посещаемых интернет-платформ: пользователи ежедневно ищут рецепты, делятся кулинарным опытом и сохраняют понравившиеся блюда. Тем не менее большинство подобных сервисов либо перегружены рекламой, либо не предоставляют достаточного уровня интерактивности. Создание собственной платформы для хранения и просмотра рецептов с возможностью оставлять отзывы и оценки представляет собой актуальную практическую задачу в области fullstack-разработки.

В рамках учебной практики по профессиональному модулю ПМ3 «Разработка модулей программного обеспечения для компьютерных систем» разработано веб-приложение «КнигаРецептов» — онлайн-платформа для просмотра кулинарных рецептов с системой отзывов, рейтингов и избранного. В качестве основной базы данных использован облачный сервис Firebase Firestore, что позволило реализовать хранение данных без необходимости развёртывания собственного серверного окружения.

Целью практики является практическое освоение технологий разработки современных веб-приложений на базе облачной платформы Firebase: проектирование структуры документно-ориентированной базы данных, реализация системы аутентификации пользователей средствами Firestore, разработка клиентского интерфейса с использованием JavaScript в модульной архитектуре, а также настройка правил безопасности и оптимизация запросов к базе данных.

Для достижения поставленной цели решены следующие задачи:

— анализ предметной области и составление технического задания на разработку веб-приложения;

— проектирование схемы базы данных Firebase Firestore с учётом требований к производительности;

— реализация системы аутентификации пользователей на основе коллекции Firestore без использования Firebase Authentication;

— разработка каталога рецептов с постраничной загрузкой, поиском и фильтрацией по категориям;

— создание страницы детального просмотра рецепта с системой отзывов, рейтингов и избранного;

— разработка личного кабинета пользователя с историей действий;

— реализация административной панели для управления контентом и пользователями;

— настройка правил безопасности Firestore и оптимизация запросов.

В ходе практики применялись следующие методы: изучение официальной документации Firebase, итеративное проектирование с пошаговым усложнением функционала, тестирование работоспособности отдельных модулей, а также анализ и отладка запросов к Firestore через консоль разработчика браузера.

Ожидаемым результатом практики является полнофункциональное веб-приложение, соответствующее всем требованиям технического задания, с задокументированной архитектурой, схемой базы данных и правилами безопасности.

1 ПОСТАНОВКА ЗАДАЧИ (ТЕХНИЧЕСКОЕ ЗАДАНИЕ)

1.1 Цель проекта

Целью проекта является разработка полнофункционального веб-приложения «КнигаРецептов» — онлайн-платформы для публикации и просмотра кулинарных рецептов с использованием облачной базы данных Firebase Firestore в качестве единственного хранилища данных.

Приложение реализует следующие функциональные возможности: каталог рецептов с постраничной загрузкой и фильтрацией по категориям, страница детального просмотра с системой отзывов и рейтингов, личный кабинет пользователя с историей действий, система сохранения рецептов в избранное, а также административная панель для управления контентом и пользователями.

Особенностью проекта является реализация собственной системы аутентификации на основе коллекции Firestore без применения Firebase Authentication. Данный подход выбран намеренно с целью получения практических навыков работы непосредственно с документами и коллекциями облачной базы данных: создание документа пользователя при регистрации, хеширование пароля с использованием браузерного API SubtleCrypto (алгоритм SHA-256), хранение идентификатора сессии в localStorage, а также чтение роли пользователя из Firestore при каждом входе — что обеспечивает немедленное применение изменений роли администратором без перезапуска приложения.

Проект выполнен в соответствии с техническим заданием по профессиональному модулю ПМ3 и направлен на формирование компетенций fullstack-разработчика в части работы с облачными технологиями, документно-ориентированными базами данных и клиентской веб-разработкой.

1.2 Функциональные требования

Функциональные требования к веб-приложению «КнигаРецептов» определяют состав реализуемых страниц и описывают поведение системы с точки зрения конечного пользователя.

Главная страница (index.html) обеспечивает вывод каталога рецептов в виде карточек с постраничной загрузкой по 12 элементов через механизм cursor-based пагинации Firestore (метод startAfter). На странице реализованы фильтрация по категориям блюд (Завтрак, Супы, Салаты, Выпечка, Десерты, Ужины, Напитки), сортировка по дате добавления, среднему рейтингу и количеству отзывов, а также поиск по названию рецепта на стороне клиента. Новые рецепты, добавленные администратором, отображаются без перезагрузки страницы благодаря подписке onSnapshot.

Страница детального просмотра рецепта (recipe.html) содержит полную информацию о блюде: категорию, время приготовления, количество порций, уровень сложности, список ингредиентов и пошаговое описание приготовления. Средний рейтинг рецепта обновляется в режиме реального времени через onSnapshot без перезагрузки страницы. Авторизованные пользователи могут добавить рецепт в избранное и оставить отзыв с оценкой от 1 до 5 звёзд. Пересчёт среднего рейтинга производится атомарно с помощью транзакции Firestore (runTransaction).

Страница авторизации (auth.html) содержит две вкладки: вход в систему и регистрация. При регистрации в коллекции users создаётся документ с полями uid, email,

name, passwordHash, role и createdAt. Пароль хешируется на стороне клиента с использованием браузерного API SubtleCrypto по алгоритму SHA-256 с добавлением соли. В localStorage сохраняется только идентификатор пользователя (uid). Роль пользователя считывается из Firestore при каждом входе, что исключает необходимость обновления сессии при смене роли администратором.

Личный кабинет (profile.html) отображает три раздела через навигацию в боковой панели: историю отзывов пользователя из подколлекции history, список сохранённых в избранное рецептов из коллекции favorites, а также форму редактирования профиля. Доступ к личному кабинету предоставляется только авторизованным пользователям: при отсутствии активной сессии производится автоматический редирект на страницу авторизации.

Административная панель (admin.html) доступна исключительно пользователям с ролью admin. Проверка роли производится при каждом обращении к странице путём чтения документа пользователя из Firestore — без использования кешированных данных. Панель включает три раздела: управление рецептами (создание, редактирование и удаление через модальное окно), управление пользователями (просмотр списка и изменение роли одной кнопкой), а также модерацию отзывов (просмотр всех отзывов по всем рецептам и удаление с пересчётом рейтинга через транзакцию).

Таблица 1 — Функциональные требования к страницам приложения

Страница	**Функциональность**	**Доступ**
index.html	Каталог рецептов, фильтрация по 7 категориям, сортировка, поиск, пагинация, real-time обновления	Все пользователи
recipe.html	Детальный просмотр, отзывы, рейтинг в режиме реального времени, избранное	Все / авторизованные
auth.html	Регистрация и вход в систему на основе Firestore, хеширование пароля	Гости
profile.html	История отзывов, избранное, редактирование профиля	Авторизованные
admin.html	CRUD рецептов, управление пользователями, модерация отзывов	Только admin

Примечание — составлено автором на основании проведённого исследования
Данные таблицы 1 отражают полный состав страниц приложения и разграничение прав доступа по ролям пользователей.

1.3 Нефункциональные требования

Нефункциональные требования определяют качественные характеристики системы, не связанные напрямую с её функциями, однако существенно влияющие на удобство использования и надёжность приложения.

В части технического стека проект реализован на основе JavaScript с использованием модульной архитектуры ES6 (import/export). HTML-разметка соответствует стандарту HTML5, стилевое оформление выполнено на CSS3 с применением адаптивной вёрстки на основе Flexbox и Grid. Для взаимодействия с базой

данных применяется Firebase Firestore SDK версии 10.12.0, подключаемый через CDN без дополнительных систем сборки. Каждой HTML-странице соответствует отдельный JavaScript-модуль; общая логика (аутентификация, сессия, утилиты) вынесена в файл app.js.

Требования к производительности включают: применение постраничной загрузки через startAfter для предотвращения единовременного получения всего набора данных; использование denormalization — дублирование названия рецепта в документах коллекций favorites и подколлекции history для исключения дополнительных запросов при отображении списков; предварительное вычисление агрегатных значений avgRating и ratingCount непосредственно в документах коллекции recipes с обновлением через транзакцию при каждом новом отзыве.

Требования безопасности предусматривают: проверку роли пользователя путём чтения документа из Firestore перед предоставлением доступа к административным функциям; хранение в localStorage исключительно идентификатора сессии без конфиденциальных данных; хеширование пароля на стороне клиента перед записью в базу данных; применение правил безопасности Firestore, ограничивающих операции записи в соответствии с ролями пользователей.

С точки зрения удобства использования приложение поддерживает адаптивную вёрстку для устройств с шириной экрана от 320 пикселей с тремя контрольными точками (480, 768 пикселей), отображает визуальные индикаторы загрузки при выполнении запросов к Firestore, а также информирует пользователя о результатах операций через всплывающие toast-уведомления.

1.4 Критерии готовности

Критерии готовности определяют условия, при соблюдении которых проект считается реализованным в полном объёме в соответствии с требованиями технического задания по модулю ПМ3.

К обязательным критериям относятся следующие:

— система аутентификации на основе Firestore функционирует корректно: регистрация создаёт документ в коллекции users, вход проверяет хеш пароля, сессия сохраняется между перезагрузками страницы;

— каталог рецептов отображается с постраничной загрузкой через startAfter, реализованы фильтрация по категориям и сортировка по нескольким полям;

— система отзывов и рейтингов функционирует: транзакция runTransaction атомарно обновляет поля avgRating и ratingCount в документе коллекции recipes;

— личный кабинет отображает историю отзывов из подколлекции history и список избранных рецептов из коллекции favorites;

— схема базы данных задокументирована: описаны все коллекции (users, recipes, favorites), поля документов с типами данных и связи между коллекциями;

— правила безопасности Firestore настроены и загружены в Firebase Console.

К продвинутым критериям, выполнение которых обеспечивает максимальный балл, относятся:

— режим реального времени реализован в модуле recipe.js: рейтинг рецепта обновляется через onSnapshot без перезагрузки страницы;

— административная панель содержит все три раздела: управление рецептами, пользователями и модерацию отзывов;

— запросы к Firestore оптимизированы: применяется денормализация данных и предварительное вычисление счётчиков;

— код организован по модульному принципу: семь JavaScript-модулей, общая логика вынесена в app.js.

Таблица 2 — Соответствие реализованного функционала критериям оценки

Критерий оценки	**Максимальный балл**	**Статус**
Настройка Firebase и аутентификация	10	Реализовано
Работа с Firestore (CRUD, запросы)	10	Реализовано
Реализация основных функций по теме	10	Реализовано
Пагинация и поиск/фильтрация	10	Реализовано
Админ-панель и правила безопасности	10	Реализовано
Модульная структура, сервисы	5	Реализовано
Обработка ошибок и граничных случаев	5	Реализовано
Оптимизация запросов (денормализация)	5	Реализовано
Демонстрация работающего приложения	5	Планируется на защите
Ответы на вопросы, понимание архитектуры	5	Планируется на защите

Примечание — составлено автором на основании проведённого исследования
Приведённые в таблице 2 данные свидетельствуют о том, что все обязательные критерии выполнены в полном объёме. Критерии, связанные с защитой отчёта, будут реализованы на завершающем этапе практики.

2 ПРОЕКТИРОВАНИЕ САЙТА

2.1 Проектирование базы данных

Для хранения данных веб-приложения «КнигаРецептов» использована документно-ориентированная база данных Firebase Firestore. В отличие от реляционных баз данных, Firestore организует данные в виде коллекций, содержащих документы с произвольными полями. Это позволяет гибко моделировать структуру данных и обеспечивает горизонтальное масштабирование без необходимости администрирования сервера.

При проектировании схемы базы данных применялся принцип денормализации: часть данных намеренно дублируется в нескольких коллекциях с целью минимизации числа запросов при отображении списков. Так, название рецепта (recipeTitle) хранится как в основном документе коллекции recipes, так и в документах коллекций favorites и подколлекции history, что позволяет отображать списки избранного и историю действий без дополнительных обращений к Firestore.

База данных состоит из трёх основных коллекций верхнего уровня и двух подколлекций.

Коллекция users предназначена для хранения данных зарегистрированных пользователей. Идентификатор документа генерируется на стороне клиента при регистрации в формате «u» + timestamp + случайная строка. Документ содержит следующие поля: uid (string) — уникальный идентификатор пользователя, совпадающий с идентификатором документа; email (string) — адрес электронной почты в нижнем регистре, используется для поиска при входе; name (string) — отображаемое имя пользователя; passwordHash (string) — хеш пароля, вычисленный алгоритмом SHA-256 с солью на стороне клиента; role (string) — роль пользователя, принимает значения «user» или «admin»; createdAt (timestamp) — дата и время регистрации, устанавливается функцией serverTimestamp().

Внутри каждого документа коллекции users находится подколлекция history, предназначенная для хранения истории действий пользователя. Каждый документ подколлекции содержит: type (string) — тип действия (значение «review»); recipeId (string) — идентификатор рецепта; recipeTitle (string) — название рецепта (денормализованное поле); rating (number) — оценка от 1 до 5; text (string) — текст отзыва; createdAt (timestamp) — дата и время создания.

Коллекция recipes является центральной коллекцией приложения и содержит данные о рецептах. Идентификатор документа генерируется автоматически функцией addDoc. Документ содержит следующие поля: title (string) — название рецепта; category (string) — категория из фиксированного списка (Завтрак, Супы, Салаты, Выпечка, Десерты, Ужины, Напитки); description (string) — краткое описание блюда; ingredients (array of string) — массив ингредиентов, каждый элемент соответствует одной строке; steps (array of string) — массив шагов приготовления; cookTime (number) — время приготовления в минутах; servings (number) — количество порций; difficulty (string) — уровень сложности (Легко, Средне, Сложно); avgRating (number) — средний рейтинг, пересчитывается транзакцией при каждом новом отзыве; ratingCount (number) — количество отзывов, используется совместно с avgRating для пересчёта среднего значения; authorId (string) — идентификатор автора (администратора); authorName (string) — имя автора (денормализованное поле); createdAt (timestamp) — дата создания; updatedAt (timestamp) — дата последнего обновления.

Внутри каждого документа коллекции recipes находится подколлекция reviews, содержащая отзывы пользователей. Каждый документ подколлекции содержит: userId

(string) — идентификатор автора отзыва; authorName (string) — имя автора (денормализованное поле); rating (number) — оценка от 1 до 5; text (string) — текст отзыва; createdAt (timestamp) — дата создания; updatedAt (timestamp) — дата последнего редактирования.

Коллекция favorites предназначена для хранения записей об избранных рецептах пользователей. Идентификатор документа формируется по составному принципу: «{userId}_{recipeId}», что исключает дублирование записей и позволяет напрямую обращаться к документу без дополнительного запроса. Документ содержит: userId (string) — идентификатор пользователя; recipeId (string) — идентификатор рецепта; recipeTitle (string) — название рецепта (денормализованное поле); category (string) — категория рецепта (денормализованное поле); savedAt (timestamp) — дата добавления в избранное.

Таблица 3 — Структура коллекций базы данных Firestore

Коллекция	**Идентификатор документа**	**Основные поля**	**Связи**
users	uid (генерируется клиентом)	uid, email, name, passwordHash, role, createdAt	Подколлекция history
recipes	auto (addDoc)	title, category, ingredients[], steps[], avgRating, ratingCount, authorId, createdAt	Подколлекция reviews
favorites	{userId} _{recipeId}	userId, recipeId, recipeTitle, category, savedAt	Ссылается на users и recipes
users/history	auto (addDoc)	type, recipeId, recipeTitle, rating, text, createdAt	Подколлекция users
recipes/ reviews	auto (addDoc)	userId, authorName, rating, text, createdAt, updatedAt	Подколлекция recipes

Примечание — составлено автором на основании проведённого исследования

Структура базы данных, представленная в таблице 3, обеспечивает баланс между нормализацией и производительностью: критичные для отображения поля денормализованы, агрегатные значения вычисляются заблаговременно, а составные идентификаторы документов исключают дублирование в коллекции favorites.

2.2 Внешняя структура сайта

Внешняя структура сайта отражает логику навигации между страницами приложения с точки зрения конечного пользователя. Приложение «КнигаРецептов» состоит из пяти HTML-страниц, каждая из которых выполняет самостоятельную функцию и связана с остальными через навигационные элементы интерфейса.

Главная страница (index.html) является точкой входа в приложение и доступна всем пользователям без авторизации. На странице отображается каталог рецептов в виде карточек с фильтрами по категориям и элементами управления сортировкой. Из главной страницы пользователь может перейти на страницу детального просмотра любого рецепта, на страницу авторизации, а также в личный кабинет (при наличии активной сессии).

Страница детального просмотра рецепта (recipe.html) открывается при нажатии на карточку рецепта и получает идентификатор рецепта из параметра URL (id). На странице доступны переход в личный кабинет, возврат на главную страницу через навигационную панель, а также добавление рецепта в избранное и публикация отзыва (только для авторизованных пользователей).

Страница авторизации (auth.html) содержит две вкладки: вход и регистрацию. После успешной авторизации пользователь автоматически перенаправляется на главную страницу. Если пользователь уже авторизован и открывает auth.html напрямую, происходит автоматический редирект на index.html.

Личный кабинет (profile.html) доступен только авторизованным пользователям. При попытке неавторизованного пользователя открыть страницу происходит редирект на auth.html. Из личного кабинета доступны переходы на страницы детального просмотра рецептов из истории и избранного, а также ссылка на административную панель (отображается только для пользователей с ролью admin).

Административная панель (admin.html) доступна только пользователям с ролью admin. При попытке пользователя без соответствующей роли открыть страницу происходит редирект на index.html. Панель не имеет навигационных переходов на другие страницы приложения, кроме ссылки «На сайт», возвращающей пользователя на главную страницу.

Таблица 4 — Внешняя структура сайта

Страница	**Основное назначение**	**Доступные переходы**	**Доступ**
index.html	Каталог рецептов	ecipe.html, auth.html, profile.html	Все пользователи
recipe.html	Детальный просмотр рецепта	index.html, profile.html, auth.html	Все пользователи
auth.html	Вход и регистрация	index.html (после входа)	Только гости
profile.html	Личный кабинет	recipe.html, admin.html (для admin)	Авторизованные
admin.html	Управление контентом	index.html	Только admin

Примечание — составлено автором на основании проведённого исследования
Данные таблицы 4 отражают логику навигации между страницами приложения и разграничение доступа в зависимости от роли пользователя.

2.3 Внутренняя структура сайта

Внутренняя структура сайта описывает состав файлов и папок проекта, а также назначение каждого модуля и характер взаимодействия между ними.
Проект организован в виде плоской файловой структуры с двумя поддиректориями: css для стилей и js для JavaScript-модулей. Все HTML-страницы расположены в корневом каталоге проекта.

Корневой каталог содержит следующие HTML-файлы: index.html — главная страница с каталогом рецептов; recipe.html — страница детального просмотра рецепта; auth.html — страница авторизации и регистрации; profile.html — личный кабинет

пользователя; admin.html — административная панель. Помимо HTML-файлов в корневом каталоге расположен файл firestore.rules, содержащий правила безопасности Firestore.

Директория css содержит единственный файл style.css, в котором сосредоточены все стили приложения. Файл организован по принципу разделения на секции: переменные CSS (цветовая схема, отступы, шрифты), глобальные стили, компоненты шапки и навигации, карточки рецептов, страница детального просмотра, формы авторизации, профиль пользователя, административная панель, модальное окно, toast-уведомления и адаптивные стили.

Директория js содержит семь JavaScript-модулей. Файл firebase-config.js выполняет инициализацию приложения Firebase и экспортирует объект db — экземпляр Firestore. Данный модуль импортируется всеми остальными модулями, работающими с базой данных. Файл app.js является ядром приложения и содержит всю логику, не привязанную к конкретной странице: управление сессией (getSessionUid, setSessionUid, clearSession), хеширование пароля (hashPassword), загрузку данных текущего пользователя из Firestore (loadCurrentUser), функции регистрации (register) и входа (login), выхода из системы (logout), вспомогательные функции проверки роли (isAdmin), редиректов (requireAuth, requireAdmin), рендера шапки (renderHeader), а также утилиты отображения (showToast, renderStars, categoryEmoji, plural).

Файл auth.js содержит логику страницы авторизации: переключение между вкладками входа и регистрации, обработку отправки формы и отображение ошибок. Файл index.js реализует логику главной страницы: загрузку рецептов из Firestore с пагинацией, фильтрацию по категориям, сортировку, поиск на стороне клиента и подписку onSnapshot для отслеживания новых рецептов. Файл recipe.js реализует логику страницы детального просмотра: загрузку данных рецепта, подписку onSnapshot для обновления рейтинга в реальном времени, управление избранным, форму отзыва со звёздочным рейтингом и список отзывов с пагинацией. Файл profile.js реализует логику личного кабинета: загрузку истории отзывов из подколлекции history, загрузку избранного из коллекции favorites и форму редактирования профиля. Файл admin.js реализует логику административной панели: CRUD-операции с рецептами через модальное окно, управление ролями пользователей и модерацию отзывов.

Таблица 5 — Внутренняя структура сайта

Файл	**Назначение**	**Импортирует**
firebase-config.js	Инициализация Firebase, экспорт db	—
app.js	Сессия, аутентификация, утилиты	firebase-config.js
auth.js	Логика страницы auth.html	app.js
index.js	Логика страницы index.html	firebase-config.js, app.js
recipe.js	Логика страницы recipe.html	firebase-config.js, app.js
profile.js	Логика страницы profile.html	firebase-config.js, app.js
admin.js	Логика страницы admin.html	firebase-config.js, app.js

Примечание — составлено автором на основании проведённого исследования

Данные таблицы 5 отражают модульную архитектуру проекта: каждый JS-файл отвечает за строго определённую область, что обеспечивает читаемость кода и возможность независимого тестирования отдельных модулей.

3 ПРАКТИЧЕСКАЯ ЧАСТЬ РЕАЛИЗАЦИИ ПРОЕКТА

3.1 Настройка Firebase проекта

Первым этапом реализации проекта является создание и настройка проекта в Firebase Console. Для этого необходимо перейти на сайт console.firebase.google.com, создать новый проект и перейти в раздел настроек проекта.

В разделе «Project Settings» на вкладке «General» создано веб-приложение путём нажатия кнопки с иконкой «</>». После регистрации приложения Firebase Console предоставляет объект конфигурации, содержащий следующие поля: apiKey, authDomain, projectId, storageBucket, messagingSenderId и appId. Данные значения скопированы в файл js/firebase-config.js проекта.

В разделе «Firestore Database» создана база данных в тестовом режиме (Start in test mode), что позволяет выполнять операции чтения и записи без ограничений на начальном этапе разработки. В качестве региона размещения базы данных выбран ближайший доступный регион. После создания базы данных в раздел «Rules» загружены правила безопасности из файла firestore.rules проекта.

Файл js/firebase-config.js инициализирует приложение Firebase и экспортирует объект db для использования во всех остальных модулях. Импорт Firebase SDK выполняется через CDN без установки дополнительных пакетов:

```
import { initializeApp } from "https://www.gstatic.com/firebasejs/10.12.0/firebase-app.js";
import { getFirestore } from "https://www.gstatic.com/firebasejs/10.12.0/firebase-firestore.js";
```

Важным условием корректной работы приложения является запуск через локальный HTTP-сервер, а не путём прямого открытия HTML-файлов в браузере. Это обусловлено тем, что ES6-модули (import/export) не работают по протоколу file://. Для запуска использована команда python -m http.server 8000, после чего приложение становится доступным по адресу http://localhost:8000.

3.2 Создание схемы базы данных и тестовых данных

После настройки Firebase проекта и подключения Firestore выполнено создание тестовых данных через Firebase Console для проверки корректности работы всех модулей приложения.

В разделе «Firestore Database» консоли созданы тестовые документы в коллекции recipes. Каждый документ содержит все поля, предусмотренные схемой базы данных: title, category, description, ingredients (массив строк), steps (массив строк), cookTime, servings, difficulty, avgRating (начальное значение 0), ratingCount (начальное значение 0), authorName и createdAt. Заполнение тестовыми данными позволило проверить корректность отображения карточек на главной странице и содержимого на странице детального просмотра до реализации административной панели.

Для коллекции users документы создаются автоматически в процессе регистрации через форму на странице auth.html. Ручное создание документов в коллекции users не требуется, однако для первоначального назначения роли администратора необходимо после регистрации открыть Firebase Console, найти созданный документ по полю email и изменить значение поля role с «user» на «admin».

Для обеспечения работы запросов с фильтрацией и сортировкой одновременно по нескольким полям в Firebase Console настроены составные индексы. В частности, для запроса с условием where('category', '==', значение) и orderBy('createdAt', 'desc') создан составной индекс по полям category (по возрастанию) и createdAt (по убыванию). Аналогичные индексы созданы для сочетаний с полями avgRating и ratingCount.

Правила безопасности, загруженные в раздел «Rules» консоли Firestore, обеспечивают открытый доступ на чтение и запись для всех операций на период разработки. Перед развёртыванием приложения в производственной среде правила подлежат ужесточению в соответствии с файлом firestore.rules, ограничивающим операции записи в коллекцию users и операции управления рецептами.

3.3 Реализация системы аутентификации

Система аутентификации приложения «КнигаРецептов» реализована без использования Firebase Authentication — исключительно средствами Firestore. Вся логика аутентификации сосредоточена в файле js/app.js.

Управление сессией основано на хранении идентификатора пользователя в localStorage браузера. Функция getSessionUid возвращает сохранённый uid или null при отсутствии активной сессии. Функция setSessionUid записывает uid в localStorage, функция clearSession удаляет запись. Данный подход обеспечивает сохранение сессии между перезагрузками страницы и закрытием вкладки браузера.

Функция loadCurrentUser является ключевой функцией модуля: она читает uid из localStorage, затем выполняет запрос getDoc к документу коллекции users с данным идентификатором и возвращает полный объект пользователя включая актуальное значение поля role. Именно благодаря этому подходу изменение роли пользователя в Firebase Console вступает в силу немедленно после следующего входа в систему — без необходимости обновления токена или перезапуска приложения.

Хеширование пароля выполняется функцией hashPassword с использованием браузерного API SubtleCrypto. К паролю добавляется фиксированная соль «rb_salt_», после чего строка кодируется в байты через TextEncoder и передаётся в функцию crypto.subtle.digest с алгоритмом «SHA-256». Результат преобразуется в шестнадцатеричную строку и сохраняется в поле passwordHash документа пользователя.

Функция register принимает email, password и name. Перед созданием документа выполняется проверка уникальности email через запрос getDocs с условием where('email', '==', email). При обнаружении существующего документа выбрасывается исключение с сообщением «Этот email уже зарегистрирован». При успешной проверке генерируется уникальный идентификатор пользователя, вычисляется хеш пароля и вызывается функция setDoc для создания документа в коллекции users. После записи документа uid сохраняется в localStorage через setSessionUid.

Функция login принимает email и password. Выполняется запрос getDocs с условием where('email', '==', email) для получения документа пользователя. При отсутствии результата выбрасывается исключение «Пользователь не найден». Затем вычисляется хеш введённого пароля и сравнивается со значением поля passwordHash документа. При несовпадении выбрасывается исключение «Неверный пароль». При успешной проверке uid сохраняется в localStorage.

На каждой странице приложения (кроме auth.html) при загрузке вызывается функция loadCurrentUser. Функции requireAuth и requireAdmin расширяют данную логику: requireAuth выполняет редирект на auth.html при отсутствии авторизованного пользователя, requireAdmin дополнительно проверяет значение поля role и выполняет редирект на index.html для пользователей без роли admin.

3.4 Разработка главной страницы с пагинацией

Главная страница приложения реализована в файле js/index.js и обеспечивает отображение каталога рецептов с постраничной загрузкой, фильтрацией, сортировкой и поиском.

Постраничная загрузка реализована через механизм cursor-based пагинации Firestore. Размер страницы составляет 12 карточек (константа PAGE = 12). При первой загрузке и при изменении фильтров выполняется запрос с методом limit(PAGE). Последний полученный документ сохраняется в переменной lastDoc. При нажатии кнопки «Показать ещё» к запросу добавляется метод startAfter(lastDoc), что обеспечивает получение следующей порции документов без повторной загрузки уже отображённых. Кнопка «Показать ещё» скрывается, если количество полученных документов меньше PAGE, что свидетельствует об отсутствии дополнительных записей.

Фильтрация по категориям реализована через добавление условия where('category', '==', activeFilter) к запросу Firestore при выборе категории, отличной от «all». Активная категория отображается визуально через CSS-класс active на соответствующей кнопке-чипе. При смене фильтра переменная lastDoc сбрасывается и выполняется новый запрос с начала.

Сортировка реализована через изменение поля в методе orderBy в зависимости от выбранного значения в элементе select. Доступные варианты сортировки: по дате добавления (createdAt, desc), по среднему рейтингу (avgRating, desc), по количеству отзывов (ratingCount, desc) и по времени приготовления (cookTime, desc).

Поиск по названию рецепта реализован на стороне клиента. При вводе текста в поле поиска с задержкой 250 миллисекунд (debounce) все карточки, уже загруженные в DOM, проверяются на соответствие поисковому запросу через атрибут data-title. Несоответствующие карточки скрываются через атрибут hidden. Данный подход избран вместо серверного поиска Firestore ввиду ограниченности возможностей полнотекстового поиска в Firestore.

Режим реального времени на главной странице реализован через подписку onSnapshot на последний добавленный документ коллекции recipes. При появлении нового документа пользователь получает toast-уведомление «Появился новый рецепт». Первое срабатывание подписки игнорируется через флаг first, чтобы избежать уведомления при начальной загрузке страницы.

3.5 Реализация поиска и фильтрации

Механизм поиска и фильтрации является ключевым элементом удобства использования каталога рецептов. В приложении «КнигаРецептов» реализовано сочетание серверной фильтрации средствами Firestore и клиентского поиска на стороне браузера.

Серверная фильтрация по категориям выполняется путём добавления метода where к запросу Firestore до его выполнения. При выборе пользователем одной из семи категорий (Завтрак, Супы, Салаты, Выпечка, Десерты, Ужины, Напитки) переменная activeFilter принимает соответствующее значение, запрос пересобирается и выполняется заново с начала. При выборе категории «Все» условие where не добавляется, что обеспечивает получение документов из всех категорий. Серверная фильтрация минимизирует объём передаваемых данных: клиент получает только те документы, которые соответствуют выбранной категории.

Сочетание фильтра по категории и сортировки в одном запросе Firestore требует наличия составного индекса. При попытке выполнить такой запрос без индекса Firestore возвращает ошибку с прямой ссылкой на создание необходимого индекса в Firebase

Console. Индексы созданы для всех используемых комбинаций полей фильтрации и сортировки.

Клиентский поиск по названию реализован без дополнительных запросов к Firestore. При формировании карточки каждого рецепта её заголовок в нижнем регистре сохраняется в атрибуте data-title соответствующего DOM-элемента. При вводе текста в поле поиска функция applySearch перебирает все карточки в DOM и устанавливает атрибут hidden для тех, чей data-title не содержит введённую строку. Поиск нечувствителен к регистру, так как и поисковый запрос, и значение data-title приводятся к нижнему регистру. Для снижения нагрузки на DOM применяется задержка 250 миллисекунд между последним нажатием клавиши и выполнением поиска.

Элемент выбора сортировки (select) позволяет пользователю изменить порядок отображения рецептов. При смене значения сортировки переменная activeSort обновляется, переменная lastDoc сбрасывается и выполняется новый запрос к Firestore с новым порядком сортировки. Сортировка применяется совместно с активным фильтром по категории.

3.6 Страница детального просмотра

Страница детального просмотра рецепта реализована в файле js/recipe.js и обеспечивает отображение полной информации о рецепте, управление избранным, систему отзывов и рейтингов.

Идентификатор рецепта извлекается из строки запроса URL с помощью объекта URLSearchParams: new URLSearchParams(location.search).get('id'). При отсутствии параметра id или при попытке загрузить несуществующий документ выполняется редирект на главную страницу.

Данные рецепта загружаются функцией getDoc по полученному идентификатору. После успешной загрузки функция renderRecipe формирует HTML-разметку страницы: выводит категорию, название, эмодзи категории в качестве визуального элемента, информационную панель с временем приготовления, количеством порций, сложностью и именем автора, описание, список ингредиентов и пронумерованные шаги приготовления.

Рейтинг рецепта отображается в режиме реального времени через подписку onSnapshot на документ рецепта. При каждом изменении полей avgRating или ratingCount блок рейтинга обновляется без перезагрузки страницы. Это позволяет пользователю видеть актуальный рейтинг сразу после публикации отзыва другим пользователем.

Управление избранным реализовано через проверку существования документа в коллекции favorites с составным идентификатором «{uid}_{recipeId}». При загрузке страницы функция watchFavorite выполняет запрос getDoc и устанавливает соответствующий текст кнопки. При нажатии кнопки функция toggleFavorite проверяет существование документа: если документ существует — удаляет его через deleteDoc, если не существует — создаёт через setDoc с заполнением всех предусмотренных полей включая денормализованное название рецепта.

Форма отзыва отображается только для авторизованных пользователей. Для неавторизованных пользователей вместо формы выводится ссылка на страницу авторизации. Выбор оценки реализован через интерактивный элемент из пяти звёздочек: при наведении курсора звёздочки подсвечиваются до выбранного значения, при нажатии значение фиксируется в переменной pickedStar. При отправке отзыва выполняется проверка: оценка должна быть выбрана, текст должен содержать не менее пяти символов.

Публикация отзыва выполняется через транзакцию Firestore (runTransaction). Внутри транзакции выполняются следующие операции атомарно: чтение текущего документа рецепта для получения актуальных значений avgRating и ratingCount; вычисление нового среднего рейтинга по формуле (avgRating * ratingCount + pickedStar) / (ratingCount + 1); создание документа в подколлекции reviews; обновление полей avgRating и ratingCount в документе рецепта; создание записи в подколлекции history пользователя. Использование транзакции гарантирует атомарность операции: при сбое любого из шагов все изменения откатываются.

3.7 Система взаимодействия (отзывы и оценки)

Система отзывов и оценок является основным механизмом взаимодействия пользователей с контентом приложения «КнигаРецептов». Отзывы хранятся в подколлекции reviews внутри каждого документа коллекции recipes, что обеспечивает логическую связь отзыва с конкретным рецептом и позволяет загружать отзывы только при открытии соответствующей страницы детального просмотра.

Загрузка отзывов реализована с постраничной пагинацией по 10 элементов (константа REV_PAGE = 10). Запрос выполняется с сортировкой по полю createdAt в убывающем порядке, что обеспечивает отображение новых отзывов первыми. При нажатии кнопки «Показать ещё» к запросу добавляется метод startAfter(lastRev), где lastRev — последний полученный документ предыдущей порции. Кнопка скрывается при получении менее 10 документов.

Каждый отзыв отображается в виде карточки, содержащей имя автора, визуальное представление оценки в виде звёздочек, дату публикации и текст отзыва. Для отзывов, опубликованных текущим авторизованным пользователем, дополнительно отображаются кнопки «Изменить» и «Удалить».

Редактирование отзыва реализовано через встроенный диалог браузера (prompt) с предзаполненным текущим текстом отзыва. После подтверждения изменений выполняется запрос updateDoc к документу в подколлекции reviews с обновлением поля text и установкой поля updatedAt через serverTimestamp. Список отзывов перезагружается после успешного обновления.

Удаление отзыва выполняется через транзакцию runTransaction аналогично публикации, но в обратном направлении: из текущего значения avgRating * ratingCount вычитается удаляемая оценка, результат делится на уменьшенное на единицу значение ratingCount. При удалении последнего отзыва поля avgRating и ratingCount устанавливаются в значение 0. Внутри той же транзакции выполняется удаление документа из подколлекции reviews через deleteDoc.

Администратор имеет возможность удалять любые отзывы через административную панель независимо от авторства. При удалении отзыва администратором пересчёт рейтинга выполняется аналогичным образом через транзакцию в файле js/admin.js.

3.8 Функциональная страница (избранное)

Функциональность избранного реализована в двух местах приложения: добавление и удаление рецепта из избранного — на странице детального просмотра (js/recipe.js), просмотр и управление списком избранного — в личном кабинете пользователя (js/profile.js).

Коллекция favorites использует составной идентификатор документа в формате «{userId}_{recipeId}». Данный подход имеет несколько преимуществ по сравнению с автоматически генерируемым идентификатором. Во-первых, исключается возможность

добавления одного рецепта в избранное дважды — повторный вызов setDoc с тем же идентификатором перезапишет существующий документ без создания дублирующей записи. Во-вторых, проверка наличия рецепта в избранном выполняется за один запрос getDoc без необходимости поиска по полям. В-третьих, удаление из избранного выполняется напрямую по известному идентификатору через deleteDoc без предварительного запроса.

При добавлении рецепта в избранное в документ записываются следующие поля: userId (идентификатор пользователя), recipeId (идентификатор рецепта), recipeTitle (название рецепта — денормализованное поле), category (категория рецепта — денормализованное поле) и savedAt (временная метка через serverTimestamp). Денормализация названия и категории рецепта позволяет отображать список избранного в личном кабинете без дополнительных запросов к коллекции recipes.

В личном кабинете список избранного загружается запросом getDocs к коллекции favorites с фильтрацией по полю userId на стороне клиента. Каждый элемент списка отображает название рецепта, дату добавления в избранное и две кнопки: «Открыть» (переход на страницу рецепта) и «Убрать» (удаление из избранного). При удалении записи из избранного список перезагружается автоматически.

3.9 Личный кабинет пользователя

Личный кабинет реализован в файле js/profile.js и доступен только авторизованным пользователям. При загрузке страницы вызывается функция requireAuth из модуля app.js: при отсутствии активной сессии пользователь перенаправляется на страницу авторизации.

Интерфейс личного кабинета организован по принципу боковой панели с навигацией. Боковая панель содержит аватар пользователя (первая буква имени на цветном фоне), имя и email пользователя, считанные из объекта сессии, а также навигационные ссылки для переключения между разделами. Для пользователей с ролью admin дополнительно отображается ссылка на административную панель.

Переключение между разделами реализовано без перезагрузки страницы: при нажатии на пункт навигации вызывается соответствующая функция загрузки данных, которая перезаписывает содержимое основной области страницы.

Раздел «Мои отзывы» загружает историю действий пользователя из подколлекции users/{uid}/history с сортировкой по полю createdAt в убывающем порядке. Каждая запись отображает название рецепта, оценку в виде звёздочек, фрагмент текста отзыва и дату публикации. Рядом с каждой записью расположены кнопки «Открыть» (переход на страницу рецепта) и «✕» (удаление записи из истории). Удаление выполняется через deleteDoc к документу подколлекции history и не затрагивает оригинальный отзыв в подколлекции reviews рецепта.

Раздел «Избранное» загружает список сохранённых рецептов из коллекции favorites, как описано в подразделе 3.8. При отсутствии записей отображается сообщение «Избранное пусто» с рекомендацией сохранять понравившиеся рецепты.

Раздел «Настройки» содержит форму редактирования профиля с полем для изменения имени пользователя и заблокированным полем email. При сохранении изменений выполняется запрос updateDoc к документу коллекции users с обновлением поля name. Поле email не подлежит изменению, поскольку оно используется для входа в систему.

При отсутствии данных в любом из разделов отображается соответствующее сообщение с эмодзи и призывом к действию, что улучшает восприятие пустых состояний интерфейса.

3.10 Реализация режима реального времени

Режим реального времени в приложении «КнигаРецептов» реализован с использованием метода onSnapshot из Firebase Firestore SDK. Данный метод устанавливает постоянное соединение с базой данных и вызывает переданную функцию обратного вызова при каждом изменении указанного документа или коллекции.

В модуле js/recipe.js реализована подписка на изменения документа рецепта через onSnapshot(doc(db, 'recipes', id), callback). При каждом изменении полей avgRating или ratingCount в документе рецепта блок рейтинга на странице перерисовывается автоматически. Это позволяет пользователю видеть актуальное значение рейтинга в момент публикации нового отзыва другим пользователем без перезагрузки страницы. Блок рейтинга отображает числовое значение среднего рейтинга с точностью до одного знака после запятой, визуальное представление в виде звёздочек и количество отзывов с правильным склонением существительного (отзыв, отзыва, отзывов) через вспомогательную функцию plural.

В модуле js/index.js реализована подписка на последний добавленный документ коллекции recipes через onSnapshot с запросом limit(1) и сортировкой по createdAt. При появлении нового документа в коллекции пользователь получает toast-уведомление «Появился новый рецепт». Первое срабатывание подписки при инициализации страницы пропускается через флаг first = true, который устанавливается в false после первого вызова. Это исключает появление уведомления при начальной загрузке страницы, когда onSnapshot возвращает текущее состояние коллекции.

Подписки onSnapshot возвращают функцию отписки (unsubscribe), вызов которой прекращает получение обновлений и освобождает ресурсы. В текущей реализации отписка не требуется, поскольку каждая страница существует в течение одного сеанса без динамической загрузки компонентов. При масштабировании приложения с применением одностраничной архитектуры (SPA) функции отписки следует вызывать при переходе между страницами.

3.11 Админ-панель

Административная панель реализована в файле js/admin.js и предоставляет полный набор инструментов для управления контентом приложения. Доступ к панели ограничен пользователями с ролью admin: при загрузке страницы вызывается функция requireAdmin, которая читает роль пользователя непосредственно из Firestore и выполняет редирект на главную страницу для всех остальных пользователей.

Интерфейс административной панели организован в виде боковой панели с тёмным фоном и основной рабочей области. Навигация между тремя разделами реализована через обработчики событий на ссылках боковой панели без перезагрузки страницы.

Раздел управления рецептами отображает таблицу всех рецептов с колонками: название, категория, рейтинг с количеством отзывов, время приготовления и кнопки действий. Таблица загружается запросом getDocs к коллекции recipes с сортировкой по полю createdAt. Кнопка «+ Добавить» и кнопка «Изменить» у каждой строки открывают модальное окно с формой.

Модальное окно содержит поля: название рецепта (обязательное), категория (выпадающий список из семи значений), описание, время приготовления, количество порций, уровень сложности, ингредиенты (текстовая область, каждый ингредиент с новой строки) и шаги приготовления (текстовая область, каждый шаг с новой строки). При сохранении содержимое текстовых областей разбивается на массивы по символу переноса строки через метод split('\n') с последующей фильтрацией пустых строк.

Открытие модального окна реализовано через добавление CSS-класса open к элементу overlay, закрытие — через удаление класса.

При создании нового рецепта выполняется запрос addDoc к коллекции recipes с добавлением полей avgRating: 0, ratingCount: 0, authorName и createdAt. При редактировании существующего рецепта выполняется запрос updateDoc с обновлением изменённых полей и установкой поля updatedAt.

Удаление рецепта выполняется после подтверждения через диалог браузера (confirm). Запрос deleteDoc удаляет документ рецепта из коллекции recipes. Отзывы в подколлекции reviews не удаляются автоматически в текущей реализации ввиду ограничений Firestore на каскадное удаление подколлекций на стороне клиента.

Раздел управления пользователями отображает таблицу всех пользователей с колонками: имя, email, роль в виде цветного бейджа, дата регистрации и кнопка изменения роли. Таблица загружается запросом getDocs к коллекции users. Кнопка изменения роли выполняет запрос updateDoc с переключением значения поля role между «user» и «admin». Поскольку роль пользователя читается из Firestore при каждом входе, изменение роли вступает в силу после следующего входа пользователя в систему без дополнительных действий со стороны администратора.

Раздел модерации отзывов загружает все отзывы по всем рецептам. Для каждого документа коллекции recipes выполняется запрос getDocs к подколлекции reviews, результаты объединяются в единый массив с добавлением полей recipeId и recipeTitle. Таблица модерации содержит колонки: название рецепта, автор отзыва, оценка, текст отзыва и кнопка удаления. Удаление выполняется через транзакцию с пересчётом рейтинга рецепта аналогично описанному в подразделе 3.7.

3.12 Оптимизация и правила безопасности

Оптимизация запросов к Firestore и настройка правил безопасности являются завершающим этапом разработки приложения, обеспечивающим его производительность и защищённость.

В части оптимизации запросов применяется принцип денормализации данных. В документах коллекции favorites хранятся поля recipeTitle и category, скопированные из документа рецепта в момент добавления в избранное. В документах подколлекции history хранится поле recipeTitle. Это позволяет отображать списки избранного и историю действий без дополнительных запросов getDoc к коллекции recipes для каждого элемента списка, что сокращает количество операций чтения и снижает задержку отображения.

Предварительное вычисление агрегатных значений реализовано через хранение полей avgRating и ratingCount непосредственно в документе рецепта с обновлением через транзакцию при каждом добавлении или удалении отзыва. Альтернативой было бы вычисление среднего рейтинга на стороне клиента путём загрузки всех отзывов при каждом отображении карточки рецепта, что повлекло бы многократное увеличение количества операций чтения.

Постраничная загрузка через cursor-based пагинацию (startAfter) ограничивает объём данных, передаваемых при каждом запросе, до 12 документов для каталога рецептов и 10 документов для списка отзывов. Это обеспечивает быструю начальную загрузку страниц вне зависимости от общего количества документов в коллекции.

Правила безопасности Firestore описаны в файле firestore.rules. В текущей реализации применяются открытые правила (allow read, write: if true) для обеспечения доступа без Firebase Authentication. Данный подход приемлем для учебного проекта, однако для производственной среды рекомендуется ужесточить правила. В частности, операции записи в коллекцию recipes следует ограничить проверкой поля role документа

пользователя, операции создания документов в коллекции users следует разрешать только при совпадении идентификатора документа с передаваемым полем uid, а операции удаления в подколлекциях reviews следует ограничить авторами соответствующих документов.

Обработка ошибок реализована во всех асинхронных функциях через конструкцию try/catch. При возникновении ошибки в операциях с Firestore пользователь получает toast-уведомление с соответствующим сообщением. Детальная информация об ошибке выводится в консоль браузера через console.error для упрощения отладки. Граничные случаи, такие как переход на страницу несуществующего рецепта, отсутствие активной сессии при обращении к защищённым страницам и попытка доступа к административной панели без роли admin, обрабатываются через редиректы на соответствующие страницы.

Таблица 6 — Применённые методы оптимизации

Метод оптимизации	**Где применяется**	**Достигаемый эффект**
Денормализация данных	Коллекции favorites и history	Сокращение числа запросов при отображении списков
Предвычисление агрегатов	Поля avgRating и ratingCount в recipes	Отображение рейтинга без загрузки всех отзывов
Cursor-based пагинация	Каталог рецептов и список отзывов	Ограничение объёма данных при каждом запросе
Составной идентификатор	Коллекция favorites	Исключение дублирования и прямой доступ по ключу
Клиентский поиск	Главная страница	Поиск без дополнительных запросов к Firestore

Примечание — составлено автором на основании проведённого исследования

Применённые методы оптимизации, представленные в таблице 6, в совокупности обеспечивают минимальное количество операций чтения при сохранении полного функционала приложения.

ЗАКЛЮЧЕНИЕ

В ходе учебной практики по профессиональному модулю ПМ3 «Разработка модулей программного обеспечения для компьютерных систем» разработано полнофункциональное веб-приложение «КнигаРецептов» — онлайн-платформа для публикации и просмотра кулинарных рецептов с системой отзывов, рейтингов и избранного на базе облачной документно-ориентированной базы данных Firebase Firestore.

В процессе выполнения практики реализованы все требования технического задания. Спроектирована и реализована схема базы данных Firestore, включающая три коллекции верхнего уровня (users, recipes, favorites) и две подколлекции (users/{uid}/history, recipes/{rid}/reviews). Разработана собственная система аутентификации на основе Firestore без использования Firebase Authentication: регистрация и вход реализованы через операции чтения и записи документов коллекции users с хешированием пароля алгоритмом SHA-256 посредством браузерного API SubtleCrypto. Реализован каталог рецептов с cursor-based пагинацией через метод startAfter, фильтрацией по семи категориям, сортировкой по четырём полям и клиентским поиском по названию. Реализована страница детального просмотра рецепта с обновлением рейтинга в режиме реального времени через onSnapshot и атомарным пересчётом avgRating и ratingCount через транзакцию runTransaction. Разработан личный кабинет пользователя с историей отзывов, списком избранного и формой редактирования профиля. Реализована административная панель с тремя разделами: управление рецептами (CRUD), управление пользователями с возможностью изменения роли и модерация отзывов.

Проект реализован в полном объёме. К числу аспектов, которые могут быть доработаны при дальнейшем развитии приложения, относятся следующие: ужесточение правил безопасности Firestore для производственной среды с ограничением операций записи на основе роли пользователя; реализация каскадного удаления подколлекции reviews при удалении рецепта через серверную функцию Firebase Cloud Functions; добавление полнотекстового поиска через интеграцию с внешним сервисом (например, Algolia); реализация системы восстановления пароля.

В ходе практики сформированы и закреплены следующие профессиональные компетенции: проектирование структуры документно-ориентированной базы данных с учётом требований к производительности и денормализацией данных; разработка клиентских веб-приложений на JavaScript с модульной архитектурой ES6; работа с Firebase Firestore SDK — выполнение CRUD-операций, составных запросов, транзакций и подписок реального времени; реализация системы аутентификации и управления сессиями без серверной части; настройка правил безопасности Firestore; применение методов оптимизации запросов к документно-ориентированным базам данных.

Цель практики достигнута в полном объёме: разработано веб-приложение, соответствующее всем требованиям технического задания, с задокументированной архитектурой, схемой базы данных и правилами безопасности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

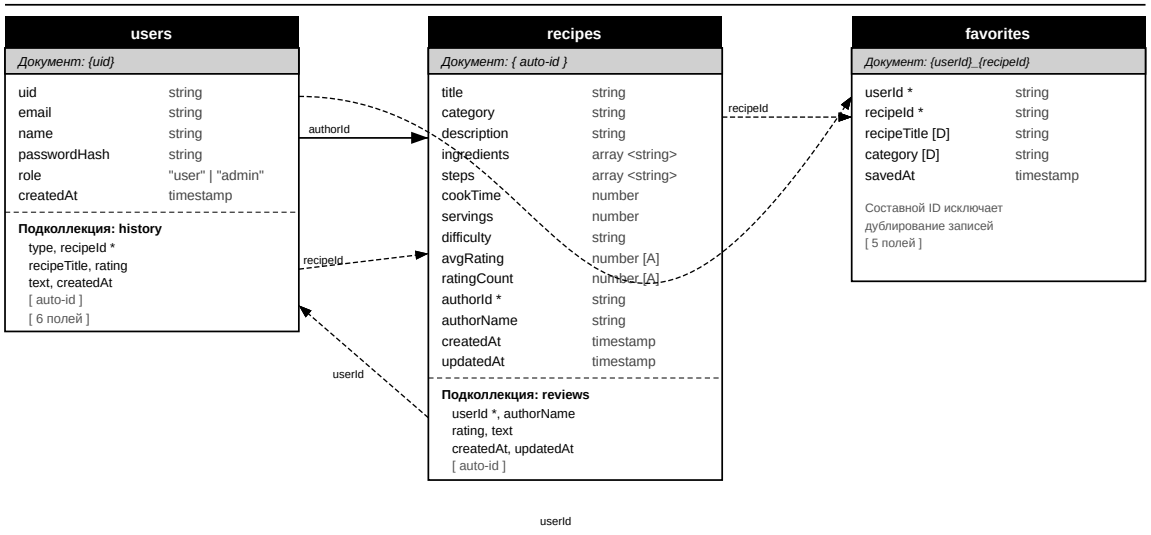
1. Google LLC. (2024). Firebase Documentation. Firebase. https://firebase.google.com/docs
2. Google LLC. (2024). Cloud Firestore Documentation. Firebase. https://firebase.google.com/docs/firestore
3. Google LLC. (2024). Get started with Cloud Firestore. Firebase. https://firebase.google.com/docs/firestore/quickstart
4. Google LLC. (2024). Structure Your Firestore Security Rules. Firebase. https://firebase.google.com/docs/firestore/security/get-started
5. Google LLC. (2024). Perform simple and compound queries in Cloud Firestore. Firebase. https://firebase.google.com/docs/firestore/query-data/queries
6. Google LLC. (2024). Paginate data with query cursors. Firebase. https://firebase.google.com/docs/firestore/query-data/query-cursors
7. Google LLC. (2024). Manage data with transactions and batched writes. Firebase. https://firebase.google.com/docs/firestore/manage-data/transactions
8. Google LLC. (2024). Get realtime updates with Cloud Firestore. Firebase. https://firebase.google.com/docs/firestore/query-data/listen
9. Mozilla Foundation. (2024). Web Crypto API — SubtleCrypto.digest(). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/digest
10. Mozilla Foundation. (2024). JavaScript modules. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules
11. Mozilla Foundation. (2024). Using Fetch. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
12. Flanagan, D. (2020). JavaScript: The Definitive Guide. 7th edition. O'Reilly Media. 706 p.
13. Google LLC. (2024). Best practices for Cloud Firestore. Firebase. https://firebase.google.com/docs/firestore/best-practices
14. Google LLC. (2024). Choose a data structure — Cloud Firestore. Firebase. https://firebase.google.com/docs/firestore/manage-data/structure-data

ПРИЛОЖЕНИЯ

Приложение 1: Схема базы данных Firestore

(Вставить схему в виде рисунка. Схема должна отображать три коллекции верхнего уровня: users, recipes, favorites. Внутри коллекции users — подколлекцию history с перечнем полей. Внутри коллекции recipes — подколлекцию reviews с перечнем полей. Для коллекции favorites указать составной идентификатор документа {userId}_{recipeId}. Связи между коллекциями обозначить стрелками: favorites ссылается на users и recipes, подколлекция history содержит поле recipeId со ссылкой на recipes.)

Схема базы данных Firebase Firestore



Обозначения:

* — поле является ссылкой на идентификатор документа другой коллекции

[D] — денормализованное поле (копируется из другой коллекции для оптимизации запросов)

[A] — агрегатное значение, пересчитывается транзакцией runTransaction при каждом отзыве

Примечание — составлено автором на основании проведённого исследования

Примечание — составлено автором на основании проведённого исследования

Приложение 2: Правила безопасности Firestore (firestore.rules)

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    function userDoc(uid) {
      return get(/databases/$(database)/documents/users/$(uid)).data;
    }
    function isAdmin(uid) {
      return userDoc(uid).role == 'admin';
    }
    function isLoggedIn() {
      // uid берём из документа — наша кастомная авторизация
      // Правила работают на уровне Firestore, поэтому
      // используем "open read" с проверкой на уровне приложения
      return true;
    }
    match /users/{uid} {
      allow read:   if true;
      allow create: if true;   // регистрация
      allow update: if true;   // обновление профиля / смена роли
      allow delete: if false;

      match /history/{hid} {
        allow read, write: if true;
      }
    }
    match /recipes/{rid} {
      allow read: if true;
      allow write: if true;   // защита через JS (проверяем роль в коде)

      match /reviews/{revId} {
        allow read:  if true;
        allow write: if true;
      }
    }
    match /favorites/{fid} {
      allow read, write: if true;
    }
  }
}
```