Санкт-Петербургский государственный университет

Кафедра Системного программирования

Группа 24.М71-мм

Разработка веб-приложения анализа повторов в документациях для проекта DocLine

Макаров Павел Михайлович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования, к.ф.-м.н., Луцив Д. В.

Санкт-Петербург
2025

Оглавление

Введение **3**

1. Постановка задачи **5**

2. Обзор **6**

2.1. Обзор предыдущей работы 6

2.2. Требования 6

2.3. Обзор новых решений 7

3. Архитектура **8**

3.1. main.go 8

3.2. Режимы поиска дубликатов 9

3.3. Утилиты 11

4. Тестирование **13**

4.1. Интеграционные тесты 13

4.2. Функциональные тесты 14

Заключение **16**

Список литературы **17**

Введение

В современном мире разработки программного обеспечения документация играет ключевую роль, обеспечивая ясность, поддерживаемость и масштабируемость проектов. Однако с ростом объёмов документации возникает проблема дублирования информации, которая усложняет её актуализацию и согласованность. Дублирование в документациях может не только привести к усложнении актуализации и согласованности, но и к появлению критических ошибок: например, при обновлении логики фреймворка разработчик мог обновить информацию в документации об объекте в одном месте, но не обновить в другом, что приведёт к недопониманию и ошибкам у пользователей фреймворка. Кроме того, при использовании данного фреймворка крупными компаниями или просто в сложной системе в критических местах, такие оплошности могут поставить под угрозу функционирование сложных информационных систем и подставить их под удар киберпреступников, охотяшихся за любой интересной для них информацией, связанной с системой или используемое ею, такими как, например, личные данные пользователей и сотрудников, информация о транзакциях, внутренняя информация компании и так далее. А при использовании фреймворка в критических системах, возможно нарушение работы таких систем, что приведёт к убытку компании. Для решения этой задачи в первом семестре был выбран проект по разработке обновлённого инструмента Duplicate Finder, предназначенного для поиска повторов в технической документации.

В предыдущем семестре основная работа была уделена созданию прототипа пользовательского интерфейса, который позволил наглядно представить концепцию будущего инструмента. Однако этот прототип обладал существенными ограничениями, такими как отсутствие полноценной реализации основных функций анализа и использование в качестве интерфейса исключительно приложение браузера.

Следующим шагом развития стало создание первой рабочей версии инструмента, написанного на языке Golang. Высокая производитель-

ность языка, простота разработки и богатая стандартная библиотека стали основными причинами выбора именно этого языка программирования в качестве основы проекта. Кроме того, одно из особенностей именно Golang - удобство создания отдельных модулей[2] - в дальнейшем упростит реализацию функций и позволит лучше контролировать и расширять архитектуру кода проекта.

1. Постановка задачи

Целью данной работы является перепроектирование и реализация инструмента Duplicate Finder (DF2) с устранением архитектурных ограничений актуальной реализации и созданием масштабируемой основы для дальнейшего развития:

1. Провести анализ существующего прототипа, выявив ограничения (архитектурные и функциональные).
2. Проанализировать функциональные и нефункциональные требования к DF2.
3. Провести обзор доступных инструментов и технологий.
4. Выработать архитектуру DF2 на основе требований и выводов обзора.
5. Реализовать сервисные компоненты DF2 на основе архитектуры.
6. Реализовать компоненты пользовательского интерфейса DF2 на основе архитектуры.
7. Провести тестирование DF2.

2. Обзор

2.1. Обзор предыдущей работы

На предыдущем этапе разработки был создан прототип веб-интерфейса, унаследовавший базовые концепции актуальной реализации инструмента Duplicate Finder. Однако данный прототип имел существенные ограничения, будучи скорее демонстрацией пользовательского интерфейса, чем полноценным рабочим инструментом. Основные недостатки заключались в реализации интерфейса пользователя в браузере, отсутствиереализации функций поиска повторов и преобладание кода разметки над кодом, реализующим функциональные компоненты проекта.

В ходе глубокого анализа текущей реализации инструмента и детального изучения ограничений существующего прототипа были сформулированы функциональные и нефункциональные требования к DF2.

2.2. Требования

Функциональные требования:

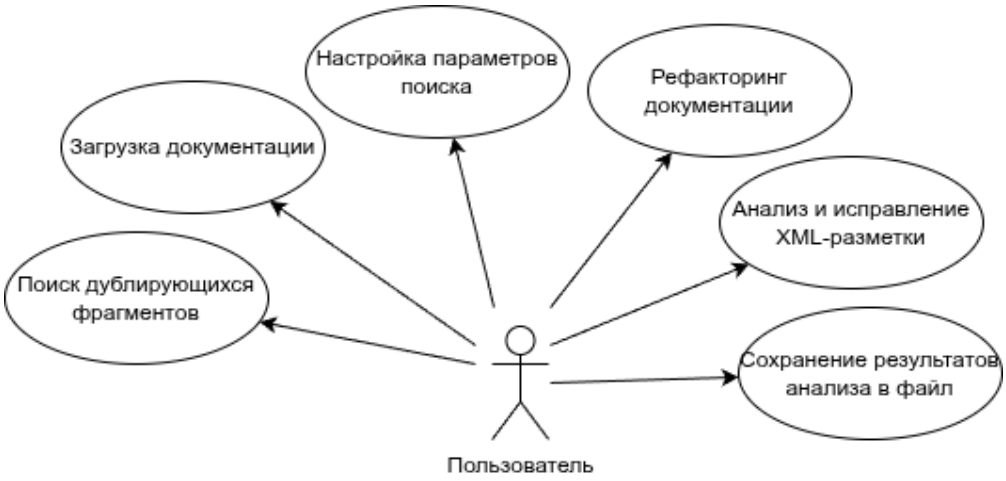

Рис. 1: Диаграмма Use Case для DF2.

- поиск дублирующих фрагментов текста в документации;
- загрузка документации для поиска повторов текста;

- настройка параметров поиска повторов текста;
- возможность рефакторинга документации;
- анализ и исправление XML-разметки;
- сохранение результатов анализа в отдельный файл.

Нефункциональные требования:

- поддержка различных форматов: DOCX, ODT, HTML, Markdown, TXT (на основе Pandoc);
- кросс-платформенность (обеспечивается Go и Go-WebUI);
- модульность: разделение на компоненты с чёткими интерфейсами (для дальнейшего добавления новых модулей).

2.3. Обзор новых решений

Одним из ключевых изменений в архитектуре проекта стал отказ от использования браузера как основы интерфейса в пользу десктопного приложения с кроссплатформенным WebView.

Для реализации графического интерфейса была выбрана библиотека Go-WebUI[1], которая позволяет использовать WebView в качестве GUI, сохраняя при этом все преимущества Go в части многопоточности и работы с системными ресурсами. Это решение потребовало отказа от ранее рассматривавшегося шаблонизатора Handlebars, так как необходимость в шаблонизаторе была пересмотрена, а также Go-WebUI предоставляет собственные механизмы взаимодействия между фронтендом и бэкендом.

Таким образом, новая архитектура сочетает преимущества веб-технологий (для интерфейса) и нативного приложения (для работы с данными), что делает инструмент более универсальным и удобным для пользователей, а также пересмотрела использование некоторых технологий в проекте. Подробнее архитектура разобрана в следующем разделе.

3. Архитектура

В рамках учебной практики была разработана первая рабочая версия приложения для поиска текстовых дубликатов в документации. Основное внимание при реализации уделялось созданию надежного рабочего ядра, способного эффективно анализировать документы различных форматов. В отличие от предыдущего прототипа, который представлял собой лишь интерфейсный макет, текущая версия реализует полный цикл обработки документов — от загрузки файлов до вывода структурированных результатов анализа. Основные слои и их взаимодействие представлены на диаграмме компонентов Рис. 2.

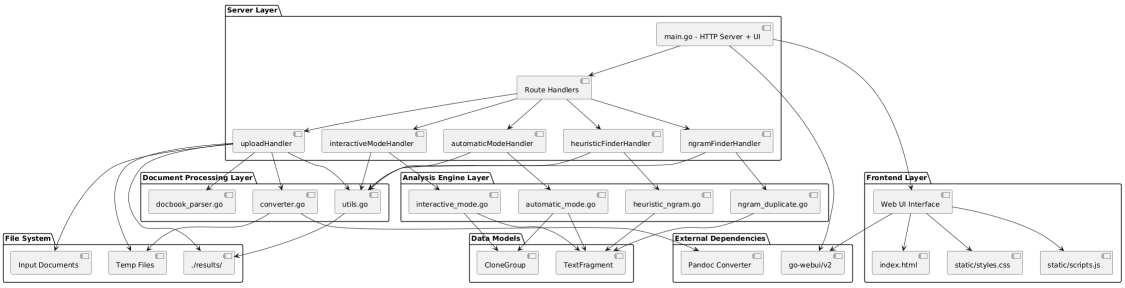

Рис. 2: Диаграмма компонентов DF2

3.1. main.go

Центральным компонентом системы является основной файл приложения, реализующий HTTP-сервер для обработки запросов. Сервер поддерживает несколько ключевых эндпоинтов, которые можно разбить на две группы по функционалу. В первой только эндпоинт /upload, который отвечает за загрузку и предварительную обработку документов: проверяется корректность формата файлов и подготавливает их к последующему анализу. Во второй – эндпоинты /automatic, /interactive, /heuristic и /ngram, которые реализуют четыре основных режима работы приложения.

Серверная часть отвечает не только за обработку запросов, но и за форматирование результатов анализа и их сохранение.

Графический интерфейс, построенный с использованием библиотеки go-webui, предоставляет удобную реализацию доступа к настройкам режимов и интегрируется с инструментом через основной файл приложения. Пример интерфейса изображён на Рис. 3.

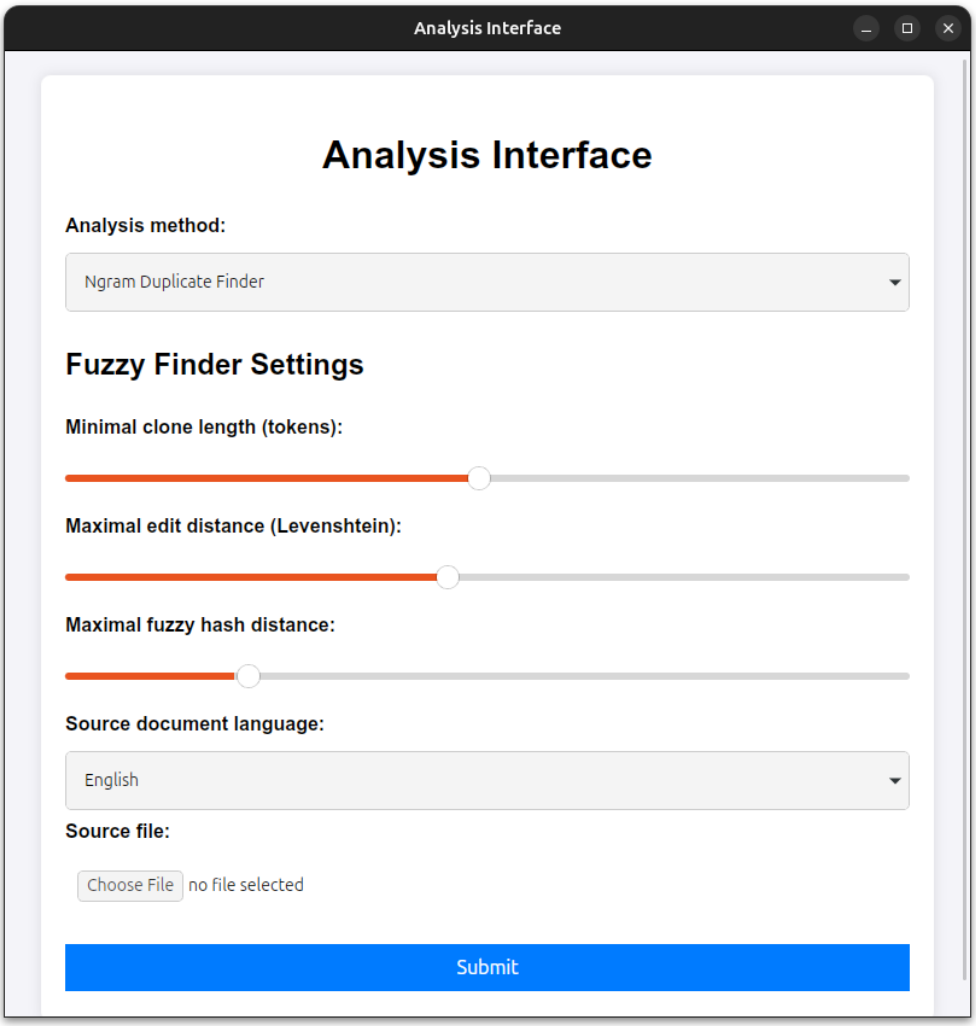

Рис. 3: Интерфейс приложения

3.2. Режимы поиска дубликатов

Реализация алгоритмов поиска повторов основана на исследованиях в области обработки естественного языка и анализе текстовых данных. Все алгоритмы разработаны на языке Go и используют единый подход к токенизации входных данных, что обеспечивает согласованность результатов при применении разных методов анализа.

Автоматический режим использует усовершенствованный алгоритм, вдохновленный принципами Clone Miner. Его ключевой особенностью является применение скользящего окна фиксированного размера для выявления схожих фрагментов текста. Алгоритм автоматически фильтрует результаты, удаляя перекрывающиеся фрагменты и выделяя наиболее значимые повторы. Особое внимание уделено оптимизации производительности, что позволяет обрабатывать большие объемы текстовых данных.

Интерактивный режим предоставляет более гибкие возможности настройки. Пользователь может задавать диапазон длин искомых фрагментов, регулировать минимальный размер групп дубликатов, включать вычисление архетипов — наиболее характерных представителей групп повторов. Алгоритм использует адаптивное скользящее окно, размер которого меняется в заданных пределах, что повышает точность поиска.

Эвристический анализ основан на обработке биграмм (последовательностей из двух слов) с применением специальных правил. Алгоритм исключает из рассмотрения фрагменты, содержащие числовые данные, и автоматически удаляет повторяющиеся n-граммы. Несмотря на относительную простоту, этот метод демонстрирует высокую эффективность при поиске точных текстовых совпадений.

N-граммный анализ с вычислением схожести по метрике Жаккара представляет собой более сложный вероятностный подход. Алгоритм строит карты n-грамм для анализируемых текстов и вычисляет степень их схожести на основе пересечения и объединения множеств n-грамм. Этот метод особенно полезен для выявления смысловых повторов и перефразированных фрагментов.

Все реализованные алгоритмы используют общие механизмы предварительной обработки текста, фильтрации результатов и постобработки. Это обеспечивает единообразие выходных данных и позволяет комбинировать результаты разных методов анализа. Особое внимание уделено оптимизации работы с памятью и многопоточной обработке данных, что критически важно для работы с большими объемами докумен-

тации.

3.3. Утилиты

Для обеспечения полноценной работы системы был разработан набор вспомогательных модулей, решающих узкоспециализированные задачи. Эти утилиты играют важную роль в подготовке данных для основного анализа и существенно расширяют возможности приложения.

Конвертер документов (converter.go) представляет собой ключевой компонент системы, отвечающий за обработку входных файлов различных форматов. Его реализация основана на модульном подходе, где для каждого поддерживаемого формата предусмотрены специализированные обработчики. Особенностью конвертера является двухэтапная обработка документов: сначала выполняется преобразование в промежуточный XML-формат, а затем — нормализация текстового содержимого. Это позволяет унифицировать последующий анализ независимо от исходного формата документа.

Система поддерживает работу с наиболее распространенными офисными форматами (DOCX, ODT), разметкой (HTML, Markdown) и простыми текстовыми файлами. Для обработки каждого формата разработаны оптимизированные алгоритмы, учитывающие специфику хранения текстовой информации. Особое внимание уделено обработке метаданных и форматирования, которые могут существенно влиять на качество поиска дубликатов.

Парсер DocBook (docbook_parser.go) реализует сложную логику извлечения текста из XML-структур. Его отличительной чертой является поддержка иерархической структуры документов — система корректно обрабатывает вложенные разделы, главы и параграфы, сохраняя при этом информацию об их взаимосвязи. Парсер включает механизмы обработки HTML-сущностей и специальных символов, что особенно важно для технической документации. Гибкая система фильтрации позволяет настраивать набор извлекаемых элементов в зависимости от конкретной задачи.

Вспомогательные утилиты(utils.go) содержат набор функций общего назначения, используемых различными компонентами системы, а именно: алгоритмы генерации n-грамм с поддержкой различных схем токенизации, функции для работы с файловой системой (включая обработку больших файлов), утилиты разбиения текста на смысловые блоки (абзацы, предложения), механизмы валидации и проверки целостности данных, инструменты логирования и отладки.

Особенностью реализации этих утилит является акцент на производительность и надежность. Например, функции работы с файлами используют буферизированное чтение/запись и поддерживают обработку данных в потоковом режиме, что критически важно для работы с объемными документами.

4. Тестирование

4.1. Интеграционные тесты

Комплекс интеграционных тестов был разработан для проверки взаимодействия всех компонентов системы в условиях

Тест TestIntegration_Endpoints выполняет сквозную **проверку всех основных API-эндпоинтов**, включая интерфейсы загрузки файлов (/upload), эвристического анализа (/heuristic), n-граммного анализа (/ngram), а также автоматического и интерактивного режимов работы. Каждый эндпоинт тестируется на корректность обработки запросов и возвращаемых статусных кодов (200 OK для успешных операций, 400 Bad Request для некорректных данных и 405 Method Not Allowed для неподдерживаемых методов).

В рамках **тестирования процедуры загрузки файлов** (TestIntegration_FileUpload) проверяются различные сценарии работы с входными данными: обработка валидных файлов, реакция на поврежденные или неподдерживаемые форматы, а также обработка специальных случаев (пустые файлы, файлы максимально допустимого размера). Для проведения тестов используется специально подготовленный набор тестовых документов, охватывающий все поддерживаемые форматы.

Тестирование методов HTTP (TestIntegration_HTTPMethods) гарантирует, что API системы корректно обрабатывает только разрешенные типы запросов. Все эндпоинты должны однозначно отклонять GET, PUT, DELETE и PATCH запросы, возвращая статус 405 Method Not Allowed. Это критически важно для безопасности системы и предотвращения непреднамеренных изменений данных.

Дополнительные тесты проверяют **корректность заголовков ответов** (TestIntegration_ResponseContentType) и **обработку ошибочных ситуаций** (TestIntegration_ErrorHandling). Первые обеспечивают, что все успешные ответы возвращаются с правильным Content-Type (application/json), что необходимо для корректной работы клиентских

приложений. Вторые проверяют устойчивость системы к некорректным запросам, включая запросы с отсутствующими или неправильными заголовками, а также обращения к несуществующим эндпоинтам.

Результаты тестов представлены в Приложении, Листинг 1.

4.2. Функциональные тесты

Функциональные тесты парсера DocBook (docbook_parser_test.go) представляют собой комплексную проверку всех аспектов работы этого критически важного компонента системы. Тесты охватывают как базовые функции, так и обработку особых случаев.

Тест инициализации (TestNewDocBookParser) проверяет корректность создания нового экземпляра парсера, включая наличие всех стандартных элементов обработки (таких как para, section, chapter). Особое внимание уделяется проверке обработки пользовательских конфигураций и параметров инициализации.

Тестирование работы с текстовыми элементами (TestDocBookParser_AddRemoveTextElement) включает проверку механизмов добавления и удаления элементов, а также обработку пользовательских элементов разметки. Тесты моделируют различные сценарии изменения структуры документа в процессе работы, гарантируя стабильность парсера при динамическом изменении входных данных.

Комплексный тест парсинга (TestDocBookParser_ParseDocBook) охватывает все основные сценарии работы с документами: обработка стандартных документов с параграфами, анализ сложных иерархических структур с вложенными элементами, корректное преобразование HTML-сущностей и специальных символов,обработка и валидация некорректного XML.

Тест извлечения текста (TestDocBookParser_ExtractText) проверяет корректность работы с различными структурными элементами документа, включая сложные случаи вложенности и комбинированного

содержимого.

Результаты тестов представлены в Приложении, Листинг 2.

Функциональные тесты конвертера (converter_test.go) обеспечивают проверку корректности преобразования документов между различными форматами. Тестирование построено по принципу "черного ящика", проверяя соответствие выходных данных ожидаемым результатам при заданных входных параметрах.

Тест определения необходимости конвертации (TestIsConversionNeeded) проверяет логику принятия решений о преобразовании форматов. Тест использует исчерпывающий набор расширений файлов, включая все поддерживаемые форматы и специальные случаи (неизвестные расширения, файлы без расширения). Это гарантирует, что система корректно определяет необходимость преобразования для каждого типа документов.

Тест определения форматов Pandoc (TestGetPandocFormat) обеспечивает правильность сопоставления между внутренним представлением форматов в системе и их идентификаторами в Pandoc. Тест проверяет обработку всех поддерживаемых типов файлов (DOCX, ODT, RTF, Markdown, TXT, HTML), а также реакцию системы на неизвестные форматы. Особое внимание уделяется обработке пограничных случаев и файлов с нестандартными расширениями.

Все функциональные тесты разработаны с учетом принципов максимального покрытия и воспроизводимости. Каждый тестовый случай включает в себя подготовку тестового окружения,генерацию входных данных, выполнение тестируемой операции, проверку результатов и очистку тестовых артефактов.

Такой подход обеспечивает надежную проверку функций системы и позволяет быстро выявлять регрессии при внесении изменений в код.

Результаты тестов представлены в Приложении, Листинг 3.

Заключение

Результатом работы стала рабочая версия инструмента Duplicate Finder (DF2), в котором реализованы:

- графический интерфейс на основе веб-технологии;
- четыре различных режимов анализа текста (автоматический, интерактивный, n-граммный и эвристический).

Направления для дальнейшей работы:

- совершенствование визуализации результатов анализа;
- оптимизация кода, устранение выявленных недостатков;
- улучшение алгоритмов анализа для повышения точности обнаружения смысловых повторов;
- изучение возможности реализации дополнительного интерфейса интерфейса командной оболочки.

Исходный код можно найти в репозитории PavelMkr/docline-new.

Список литературы

[1] Go-WebUI. — URL: https://github.com/webui-dev/go-webui (дата обращения: 1 июня 2025 г.).

[2] Golang Documentation. — URL: https://go.dev/doc/ (дата обращения: 1 июня 2025 г.).