МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные СУБД»

Тема: Хранилище исходного кода с учетом AST деревьев

Студенты гр. 2304 _____ Пикалов И.В.
_____ Бесогонов М.Р.
_____ Деменев К.О.
_____ Птицын Д.О.
_____ Жихарев В.В.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2025

ЗАДАНИЕ

Студенты группы 2304

Пикалов И.В.

Бесогонов М.Р.

Деменев К.О.

Птицын Д.О.

Жихарев В.В.

Тема работы: Хранилище исходного кода с учетом AST деревьев

Исходные данные:

Задача - создать сервис, который хостит гит репозитории с исходным кодом И обрабатывает код для конвертации его в AST дерева, чтобы делать анализ и прогнозировать будущие проблемы у разработчиков.

Полный текст задания: ИДЗ - состав, порядок работы [МОЭВМ Вики [se.moevm.info]]

Содержание пояснительной записки:

- Содержание
- Введение
- Сценарии использования
- Модель данных
- Разработанное приложение
- Заключение
- Приложения
- Литература

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 16.02.2025

Дата сдачи реферата: 27.05.2025

Дата защиты реферата: 27.05.2025

Студенты гр. 2304		Пикалов И.В.
	_____	Бесогонов М.Р.
	_____	Деменев К.О.
	_____	Птицын Д.О.
	_____	Жихарев В.В.
	_____	
Преподаватель		Заславский М.М.
	_____	

АННОТАЦИЯ

В рамках данного курса предполагалось разработать какое-либо приложение в команде на одну из поставленных тем. Была выбрана тема создания хранилища исходного кода с учетом AST деревьев. Требуется реализовать хранение данных в графовой СУБД ArangoDB. Найти исходный код и всю дополнительную информацию можно по ссылке: https://github.com/moevm/nosql1h25-asttrees

SUMMARY

As part of this course, it was supposed to develop an application in a team on one of the set topics. The topic of creating a source code repository based on AST trees was chosen. It is required to implement data storage in the ArangoDB graph database. You can find the source code and all additional information here: https://github.com/moevm/nosql1h25-asttrees

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1. Введение

1.1. Актуальность решаемой проблемы

Разработка программного обеспечения сопряжена с ростом сложности кодовых баз, что затрудняет их глубокое понимание и своевременное выявление потенциальных проблем. Традиционные инструменты просмотра кода часто ограничиваются текстовым представлением, не раскрывая его семантическую структуру. Это усложняет анализ, особенно в крупных проектах с множеством веток и изменений. Возникает потребность в инструментах, предоставляющих разработчикам и администраторам доступ к более глубокому представлению кода через Абстрактные Синтаксические Деревья (AST). Сервис, который не только хостит Git-репозитории, но и обеспечивает визуализацию AST-деревьев для различных веток и коммитов, а также предоставляет административные функции для управления и всестороннего обзора, отвечает на эту потребность, предлагая мощные возможности для детального изучения и контроля качества исходного кода.

1.2. Постановка задачи

Цель работы

Предоставление разработчикам и администраторам веб-сервиса для хостинга Git-репозиториев с функционалом автоматического построения, хранения и интерактивной визуализации AST-деревьев исходного кода, обеспечивающего детальный просмотр файлов и их структурного представления на различных ветках и коммитах, а также предоставляющего инструменты администрирования для управления пользователями и репозиториями.

Задачи

- Разработать систему аутентификации и авторизации пользователей: Реализовать регистрацию пользователей и разграничение прав доступа (пользователь/администратор).
- Реализовать модуль импорта и хостинга Git-репозиториев: Обеспечить загрузку репозиториев по URL-ссылке и их хранение на сервере.
- Разработать механизм парсинга исходного кода и построения AST-деревьев: Интегрировать или создать инструменты для автоматической генерации AST-деревьев из файлов репозитория при его загрузке.
- Реализовать хранение данных в ArangoDB: Спроектировать и внедрить структуру для хранения информации о пользователях, репозиториях, файлах, коммитах и их AST-представлениях в мультимодельной СУБД ArangoDB.
- Создать пользовательский интерфейс для просмотра кода и AST-деревьев: Разработать интуитивно понятный веб-интерфейс, позволяющий пользователям просматривать структуру файлов репозитория, их содержимое и соответствующие AST-деревья, с возможностью переключения между различными ветками и коммитами.
- Разработать панель администратора: Создать интерфейс для администратора с возможностью просмотра списка всех пользователей и репозиториев, а также детального изучения всех связанных данных (файлы, коммиты, AST-деревья) с функциями фильтрации и сортировки.
- Обеспечить навигацию и взаимодействие: Реализовать удобную навигацию по файловой структуре репозитория, коммитам и веткам, а также интерактивное отображение AST-деревьев.

1.3. Предлагаемое решение

В рамках проекта реализован специализированный сервис для хостинга Git-репозиториев, обеспечивающий углубленный анализ исходного кода через

построение и визуализацию Абстрактных Синтаксических Деревьев (AST). В качестве основы для хранения данных (информации о пользователях, репозиториях, их файловой структуре, коммитах и сгенерированных AST-деревьях) используется мультимодельная база данных ArangoDB. Это позволяет эффективно управлять как самим исходным кодом, так и его сложными структурированными представлениями в виде AST.

1.4. Качественные требования к решению

В рамках разработки сервиса для хостинга и анализа исходного кода особое внимание уделяется качественным характеристикам системы, обеспечивающим её надежную работу, удобство использования как для разработчиков, так и для администраторов, а также готовность к потенциальному расширению функционала. Ниже перечислены ключевые требования, которым соответствует реализованное решение:

- Надёжность хранения кода и его представлений: Система обеспечивает целостность данных репозиториев, их версий (коммитов) и сгенерированных AST-деревьев при операциях загрузки и просмотра.
- Интуитивно понятный пользовательский интерфейс (UI) и опыт взаимодействия (UX): Функции регистрации, загрузки репозиториев, просмотра файловой структуры, исходного кода и AST-деревьев легко доступны, логично организованы и не вызывают затруднений при использовании как у рядовых пользователей, так и у администраторов системы.
- Эффективная и интерактивная визуализация AST-деревьев: Структура исходного кода наглядно отображается в виде AST-дерева.
- Гибкость и масштабируемость архитектуры: Решение спроектировано с учетом возможности добавления поддержки новых языков программирования (для AST-парсинга), интеграции более сложных аналитических модулей (например, для прогнозирования проблем на

основе AST), а также способности справляться с ростом числа пользователей и объемов хранимых репозиториев.

- Контроль и управляемость для администраторов: Административная панель предоставляет исчерпывающие инструменты для мониторинга и управления пользователями и репозиториями, включая возможности детализированного просмотра, фильтрации и сортировки данных.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

(https://github.com/moevm/nosql1h25-asttrees/blob/main/ui_mockup.svg)

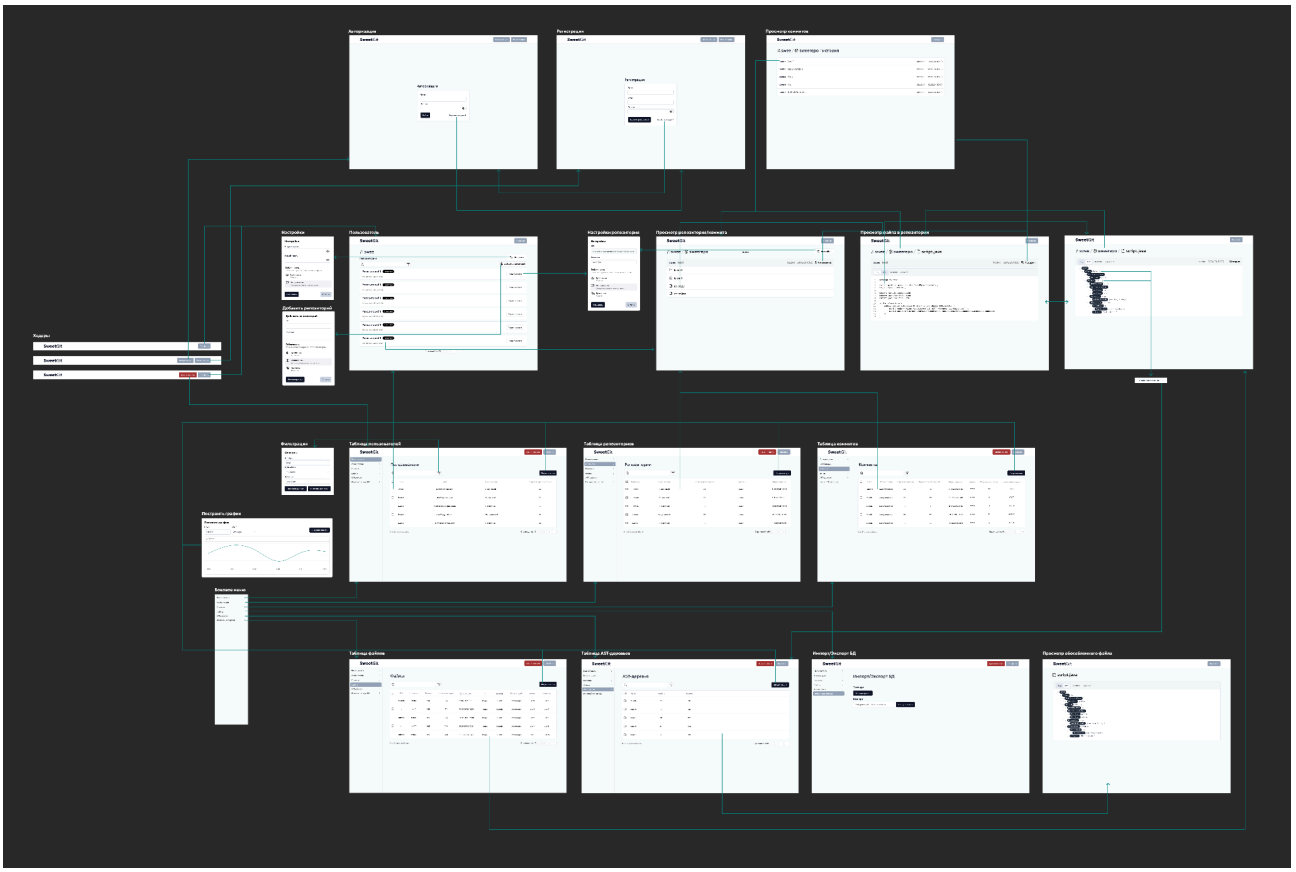

Рисунок 1 – Макет UI

2.2. Сценарии использования для различных задач

1. **Регистрация пользователя**

   Предусловие: Пользователь имеет стабильное интернет-соединение

   Основной сценарий:

   1. Пользователь открывает сервис и попадает на страницу регистрации. Отсюда ему доступны следующие поля и кнопки:
      - Поле для ввода email
      - Поле для ввода логина
      - Поле для ввода пароля
      - Кнопка "зарегистрироваться"

- Кнопка "Уже есть аккаунт"

2. После этого он вводит данные (email, пароль, логин).
3. Пользователь нажимает “зарегистрироваться”. Система сверяет данные с данными из БД, создает новый аккаунт и перенаправляет пользователя в личный кабинет на страницу “мои репозитории”

Альтернативные сценарии:

Данные некорректны

2. Пользователь вводит некорректные данные (например, пустую строку вместо пароля и т.п.). Система подсвечивает поле красным и сообщает пользователю о некорректности введенных данных

Данные уже использованы

2. Пользователь вводит данные, уже использованные при регистрации (логин или email). Система сообщает пользователю об ошибке

Переход на форму авторизации

2. Пользователь нажимает “уже есть аккаунт” и перенаправляется на страницу регистрации

2. Авторизация пользователя

Предусловие: Пользователь имеет стабильное интернет-соединение

Учетная запись пользователя существует в системе

Основной сценарий:

1. Пользователь открывает страницу авторизации, на которой представлена форма со следующими полями и кнопками:
   - Поле для ввода логина
   - Поле для ввода пароля
   - Кнопка "Войти"

- Кнопка "Еще нет аккаунта?"

2. После этого он вводит свои корректные данные (пароль и логин).
3. Пользователь нажимает “Войти”. Система сверяет данные с данными из БД и перенаправляет пользователя в личный кабинет на страницу “мои репозитории”

Альтернативные сценарии:

Данные некорректны

2. Пользователь вводит некорректные данные (например, пустую строку вместо пароля и т.п.). Система подсвечивает поле красным и сообщает пользователю о некорректности введенных данных

Данные неверны

2. Пользователь вводит неверные данные (логин или пароль неверен). Система сообщает пользователю об ошибке

Переход на форму регистрации

2. Пользователь нажимает кнопку “Еще нет аккаунта?” и перенаправляется на страницу регистрации

3. Просмотр страницы пользователя

Предусловие: Страница запрашиваемого пользователя существует и доступна просматривающему пользователю. Пользователь имеет стабильное интернет-соединение

Основной сценарий:

1. Пользователь попадает на страницу пользоваетеля. Отсюда ему доступны:
    - Основная информация о пользователе
        - По клику на кнопку "настройки" -- переход к сценарию "Настройки пользователя" (при наличии прав)

- Список репозиториев согласно настройкам публичности пользователя (с поиском, фильтрацией и пагинацией):
    - По клику на репозиторий -- переход к сценарию "Просмотр репозитория"
    - По клику на кнопку "Настройки" -- переход к сценарию "Настройки репозитория" (при наличии прав)
    - По клику на кнопку "Добавить репозиторий" -- переход к сценарию “Добавление репозитория” (при наличии прав)

Альтернативные сценарии:

Запрашиваемый пользователь не существует либо пользователь не имеет прав на просмотр его информации:

2. Отображается ошибка “страница не найдена”

4. Добавление репозитория

Предусловие: Пользователь авторизован в системе. Пользователь имеет стабильное интернет-соединение

Основной сценарий:

1. Открывается модальное окно со следующими полями и кнопками:
    - поле для ввода URL
    - поле для ввода имени репозитория
    - поле для выбора статуса публичности репозитория (публичный, защищенный, приватный)
    - кнопка “импортировать”
    - кнопка “отмена”
2. Пользователь вводит валидный URL, соответствующий публично доступному git-репозиторию, а также заполняет остальные поля.
3. Пользователь нажимает кнопку “импортировать”. Запускается процесс добавления (клонирования) ветки репозитория, анализа

коммитов и файлов и построения AST-деревьев. В результате пользователь получает уведомление об успешном добавлении репозитория и видит его в списке своих репозиториев со всей основной информацией

Альтернативные сценарии:

Данные некорректны

2. Пользователь вводит некорректные данные (например, пустую строку вместо названия или невалидный URL). Система подсвечивает поля красным и сообщает пользователю о некорректности введенных данных

Ошибка доступа к репозиторию

2. Пользователь вводит валидный URL, однако система не может получить доступ к репозиторию по ссылке

3. Пользователь нажимает “импортировать” и видит уведомление об ошибке, связанное с приватным репозиторием

Отмена добавления репозитория

2. Пользователь нажимает “отмена” (до или после заполнения данных). Введенные данные сбрасываются (очищаются поля формы), модальное окно закрывается

5. Просмотр репозитория

Предусловие: Пользователь авторизован. Пользователь имеет стабильное интернет-соединение. Запрашиваемый репозиторий существует и пользователь имеет права на его просмотр

Основной сценарий:

1. Пользователь открывает страницу просмотра репозитория. Отсюда ему доступны:

- Информация о выбранном коммите (при открытии репозитория -- о последнем)
- Информация о репозитории
- По клику на кнопку “Коммиты” -- переход к сценарию "Просмотр списка коммитов"
- По клику на файл в списке файлов -- переход к сценарию "Просмотр файла в репозитории"
- По клику на кнопку "Настройки" -- переход к сценарию "Настройки репозитория" (при наличии прав)

Альтернативные сценарии:

2. Запрашиваемый репозиторий не существует, либо пользователь не имеет прав на просмотр данного репозитория: отображается ошибка “страница не найдена”

6. Настройки пользователя

Предусловие: Пользователь авторизован. Пользователь имеет стабильное интернет-соединение.

Основной сценарий:

1. Открывается модальное окно со следующими полями и кнопками:
   - поле для ввода старого пароля
   - поле для ввода нового пароля
   - поле для выбора нового статуса публичности аккаунта (публичный, защищенный). Изначально в поле указан текущий статус публичности аккаунта
   - кнопка “сохранить”
   - кнопка “отмена”
2. Пользователь вводит корректный старый пароль, новый пароль и выбирает статус публичности аккаунта. Система проверяет валидность полей и подсвечивает красным при некорректности

3. Пользователь нажимает кнопку “сохранить”. Пароль и статус публичности данного аккаунта в системе заменяются на введенные пользователем. В результате пользователь получает уведомление об успешном изменении настроек аккаунта

Альтернативные сценарии:

Данные некорректны

2. Пользователь вводит некорректные данные (например, пустую строку вместо пароля). Система подсвечивает поля красным и сообщает пользователю о некорректности введенных данных

Старый пароль некорректен

2. Пользователь получает уведомление о том, что введенный им пароль не соответствует старому паролю

Отмена изменения настроек аккаунта

2. Пользователь нажимает “отмена” (до или после заполнения данных). Введенные данные сбрасываются (очищаются поля формы), модальное окно закрывается

7. Настройки репозитория

Предусловие: Пользователь авторизован. Пользователь имеет стабильное интернет-соединение. Запрашиваемый репозиторий существует и пользователь имеет права на его редактирование

Основной сценарий:

1. Открывается модальное окно со следующими полями и кнопками:
    - поле для ввода нового имени репозитория. Изначально в поле указано текущее имя репозитория

- поле для выбора нового статуса публичности репозитория (публичный, защищенный, приватный). Изначально в поле указан текущий статус публичности репозитория
- кнопка “изменить”
- кнопка “отмена”

2. Пользователь вводит валидное название репозитория, а также заполняет остальные поля. Система проверяет валидность полей и подсвечивает красным при некорректности.
3. Пользователь нажимает кнопку “изменить”. Название и статус публичности данного репозитория в системе заменяются на введенные пользователем. В результате пользователь получает уведомление об успешном изменении настроек репозитория и видит его в списке своих репозиториев с обновленной информацией.

Альтернативные сценарии:

Данные некорректны.

2. Пользователь вводит некорректные данные (например, пустую строку вместо названия). Система подсвечивает поля красным и сообщает пользователю о некорректности введенных данных.

Пользователь не изменил данные.

2. Пользователь не изменяет данные в полях и сразу нажимает на кнопку "изменить". Система уведомляет пользователя о том, что данные репозитория не были изменены.

Отмена изменения настроек репозитория.

2. Пользователь нажимает “отмена” (до или после заполнения данных). Введенные данные сбрасываются (очищаются поля формы), модальное окно закрывается.

8. Просмотр списка коммитов

Предусловие: Пользователь авторизован. Пользователь имеет стабильное интернет-соединение. Репозиторий, список коммитов которого рассматривается, существует и пользователь имеет права на его просмотр

Основной сценарий:

1. Пользователь открывает страницу просмотра списка коммитов репозитория. Отсюда ему доступны:
    - Список коммитов с информацией о каждом коммите
    - По клику на коммит -- переход к сценарию просмотра репозитория в момент выбранного коммита

Альтернативные сценарии:

2. Запрашиваемый репозиторий не существует, либо пользователь не имеет прав на просмотр данного репозитория: отображается ошибка “страница не найдена”

9. Просмотр файла в репозитории

Предусловие: Пользователь авторизован. Пользователь имеет стабильное интернет-соединение. Файл в запрашиваемом репозитории существует и пользователь имеет права на просмотр данного репозитория.

Основной сценарий:

1. Пользователь открывает файл для просмотра. Отсюда ему доступны:
    - Содержимое открытого файл.
    - По клику на кнопку “AST” -- вместо содержимого файла показывается AST-дерево для данного файла (при наличии сгенерированного AST-дерева для файла)
    - По клику на название репозитория, в котором находится файл -- переход к сценарию "Просмотр репозитория"

Альтернативные сценарии:

2. Запрашиваемый файл не существует, либо пользователь не имеет прав на просмотр репозитория, в котором находится файл: отображается ошибка “страница не найдена”

10. Просмотр файла

Предусловие: Пользователь авторизован. Пользователь имеет стабильное интернет-соединение. Файл существует и пользователь имеет права администратора.

Основной сценарий:

1. Пользователь открывает файл для просмотра. Отсюда ему доступны:
    - Содержимое открытого файла
    - По клику на кнопку “AST” -- вместо содержимого файла показывается AST-дерево для данного файла (при наличии сгенерированного AST-дерева для файла)

Альтернативные сценарии:

2. Запрашиваемый файл не существует, либо пользователь не имеет прав на просмотр репозитория, в котором находится файл: отображается ошибка “страница не найдена”

11. Взаимодействие с заголовком сайта

Предусловие: Пользователь авторизован. Пользователь является администратором. Пользователь имеет стабильное интернет-соединение.

Основной сценарий:

1. Находясь на любой странице сайта авторизированному пользователю в заголовке доступны кнопки "админ-панель" и "профиль"

- По клику на кнопку “админ-панель" - переход к сценарию "Просмотр таблиц в админ-панели"
- По клику на кнопку "профиль" - переход к сценарию "Просмотр страницы пользователя"

Альтернативные сценарии:

Пользователь не является администратором

1. Пользователю в заголовке доступна только кнопка "профиль"
    - По клику на кнопку "профиль" - переход к сценарию "Просмотр страницы пользователя"

Пользователь не авторизирован

1. Пользователю в заголовке доступны кнопки "авторизация" и "регистрация".
    - По клику на кнопку "авторизация" - переход к сценарию "Авторизация пользователя"
    - По клику на кнопку "регистрация" - переход к сценарию "Регистрация пользователя"

12. Просмотр таблиц в админ-панели

Предусловие: Пользователь авторизован. Пользователь является администратором. Пользователь имеет стабильное интернет-соединение.

Основной сценарий:

1. Пользователь переходит в админ панель и попадает на просмотр таблицы пользователей. Здесь же есть возможность перейти на просмотр и других страниц (репозиториев, коммитов, файлов и AST-деревьев) которые устроены тем же образом. В каждой таблице есть возможность фильтрации по любому из полей сущности, а также возможность перейти с подробному описанию сущности.
2. Действия пользователя:

- По клику на значок фильтрации в любой из таблиц открывается модальное окно с выбором настроек фильтрации (в зависимости от сущности):
    - Для фильтра доступен выбор атрибута фильтрации, типа фильтра и значения для фильтрации
    - По клику на кнопку "добавить фильтр" добавляется новый фильтр
    - По клику на кнопку "очистить фильтры" все ранее добавленные фильтры удаляются.
- По клику на кнопку "визуализация" после установленных параметров фильтрации -- переход к сценарию "Визуализация выборки"
- По клику на сущность -- переход к сценарию просмотрп данной сущности (для пользователя -- "Просмотр страницы пользователя", для репозитория/коммита -- "Просмотр репозитория", для файла/AST-дерева -- "Просмотр файла")
- По клику на "Импорт/Экспорт БД" -- переход к сценарию "Импорт/Экспорт БД"

13. Импорт/Экспорт БД

Предусловие: Пользователь авторизован. Пользователь является администратором. Пользователь имеет стабильное интернет-соединение.

Основной сценарий:

1. Пользователь переходит к странице с возможностью импорта/экспорта базы данных. Доступны кнопки "экспорт", "импорт", "выбрать файл".
2. Действия пользователя:
    - По клику на "экспорт" у пользователя начнет скачиваться файл базы данных.
    - Кнопка "импорт" изначально неактивна до выбора файла.

- По клику на "выбрать файл" открывается системное модальное окно проводника для выбора необходимого файла.
- После выбора файла при повторном клике "Выбрать файл" нововыбранный файл заменит предыдущий выбранный.
- После выбора файла кнопка "импорт" станет активной. По клику на нее данные будут добавлены в базу данных приложения и пользователь увидит уведомление об успешном добавлении.

Альтернативные сценарии:

Некорректный файл импорта:

2. Если после выбора некорректного файла пользователь нажмет "импорт", система уведомит пользователя о некорректном файле и ошибке импортирования.

Неверный формат файла импорта:

2. Если при выборе файла пользователь выберет неверный формат файла импорта, то такой файл не будет загружен, а система выдаст соответствующее уведомление.

14. Визуализация выборки

Предусловие: Пользователь авторизован. Пользователь является администратором. Пользователь имеет стабильное интернет-соединение.

Основной сценарий:

1. Пользователь открывает модальное окно визуализации для конкретной выборки сущности из условий фильтрации. Здесь ему доступны поля с атрибутами (для отображения по осям графика) и кнопка "визуализировать"
2. В каждом поле (для каждой) пользователь выбирает поле сущности для визуализации из выпадающего списка со всеми полями

сущности. При выборе атрибута в одном поле исчезает возможность выбрать этот же атрибут в другом

3. Пользователь нажимает "визуализировать". Согласно выбранным полям сущности и имеющейся выборки сущности из условий фильтрации система отображает пользователю график (статистика). При повторном изменении атрибутов в полях и повторному нажатию кнопки "визуализировать" будет показан новый график.

Альтернативный сценарий:

Поля/поле не заполнены

2. Пользователь не заполнил одно или оба поля и нажал "визуализировать". Система оповещает пользователя о незаполненных полях, подсвечивая красным и не строит график.

Сценарии использования, реализуемые в рамках прототипа "хранение и представление":

- Добавление репозитория
- Просмотр репозитория
- Взаимодействие с заголовком сайта
- Авторизация пользователя
- Просмотр таблиц в панели администрирования
- Настройки репозитория
- Настройки пользователя.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель

Модель данных реализована в мультимодельной СУБД ArangoDB и будет представлена в виде коллекций документов и коллекций рёбер. Такой подход позволяет эффективно хранить и обрабатывать сложные взаимосвязи между ключевыми сущностями системы: пользователями (users), репозиториями (repositories), их ветками (branches), коммитами (commits), файлами в этих коммитах (commit_files), а также детальными Абстрактными Синтаксическими Деревьями (ast_trees и ast_nodes) для исходного кода.

3.1.А. Графическое представление

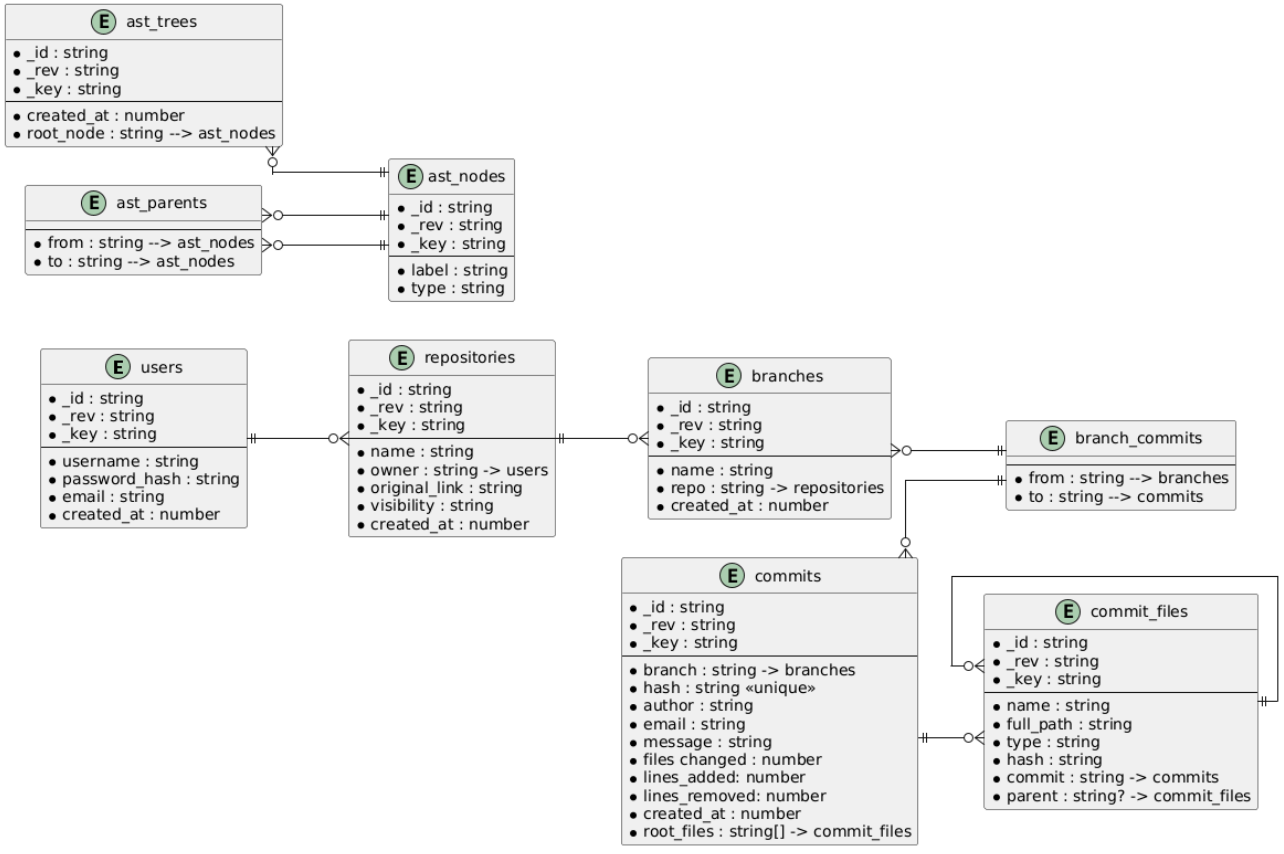

Рисунок 2 – Графическое представление нереляционной модели

3.1.B. Описание коллекций.

Размеры типов данных

- string: 4 (length) + 4 * StrSize
- number: 8
- type[]: 4 (length) + Size(type) * ArraySize

users (хранение пользователей):

- _id: string
- _rev: string
- _key: string
- username: string
- password_hash: string
- email: string
- created_at: number

repositories (хранение репозиториев):

- _id: string
- _rev: string
- _key: string
- name: string
- owner: string [-> users]
- original_link: stringы
- visibility: string
- created_at: number

branches (хранение веток):

- _id: string
- _rev: string
- _key: string
- name: string

- repo: string [-> repositories]
- created_at: number

branch_commits (связь веток и коммитов):

- _from: string
- _to: string

commits (хранение коммитов):

- _id: string
- _rev: string
- _key: string
- hash: string <<unique>>
- author: string
- email: string
- message: string
- files_changed: number
- lines_added: number
- lines_removed: number
- created_at: number
- root_files: string[] [-> commit_files]

commit_files (хранение файлов в коммите):

- _id: string
- _rev: string
- _key: string
- name: string
- full_path: string
- type: string
- hash: string # file hash
- commit: string [-> commits]

- parent: string? [-> commit_files]

ast_trees (хранение AST-деревьев по хэшу файлов):

- _id: string
- _rev: string
- _key: string # file hash
- created_at: number
- root_node: string [-> ast_nodes]

ast_nodes (хранение узлов AST-деревьев):

- _id: string
- _rev: string
- _key: string
- label: string
- type: string

ast_parents (ссылки в AST-деревьях):

- _from: string
- _to: string

3.1.C. Оценка объема информации, хранимой в модели

Примем следующие константы:

- AvgStringLength: 9
- AvgRootFilesPerCommit: 20

Получим следующие формулы для размера моделей:

- users: 40 * 6 + 8 = 248
- repositories: 40 * 7 + 8 = 288
- branches: 40 * 5 + 8 = 208
- branch_commits: 40 * 2 = 80

- commits: 40 * 7 + 8 * 4 + 20 * 40 = 1112
- commit_files: 40 * 8 = 320
- ast_trees: 40 * 4 + 8 = 168
- ast_nodes: 40 * 5 = 200
- ast_parents: 40 * 2 = 80

Оценим общий размер БД:

```
CountRepos = CountUsers * ReposPerUser
CountBranches = CountRepos * BranchesPerRepo
CountCommits = CountBranches * CommitsPerBranch
CountCommitFiles = CountCommits * FilesPerCommit
CountCommitBranches = CountCommits * BranchesPerCommit
# AST-деревья дедублицируются по ключу и не создаются для всех файлов в репозитории
CountAstTrees = CountRepos * UniqueSupportedFilesPerRepo
CountAstNodes = CountAstTrees * AstNodesPerTree

S = CountUsers * (248 + ArangoModelOverhead)
      + CountRepos * (288 + ArangoModelOverhead)
      + CountBranches * (208 + ArangoModelOverhead)
      + CountCommits * (1112 + ArangoModelOverhead)
      + CountCommitFiles * (240 + ArangoModelOverhead)
      + CountAstTrees * (168 + ArangoModelOverhead)
      + CountAstNodes * (200 + ArangoModelOverhead)
      + CountCommitBranches * (80 + ArangoModelOverhead)
      + (CountAstNodes - CountAstTrees) * (80 + ArangoModelOverhead)
      + ArangoCollectionOverhead * 9
```

Оценим зависимость размера от числа пользователей, приняв следующие константы:

```
CountUsers = X
ReposPerUser = 5
BranchesPerRepo = 5
CommitsPerBranch = 20
FilesPerCommit = 50
UniqueSupportedFilesPerRepo = BranchesPerRepo * CommitsPerBranch * FilesPerCommit / 10
ArangoCollectionOverhead = 2 * 1024 * 1024
ArangoModelOverhead = 8
AstNodesPerTree = 100
BranchesPerCommit = 2

=> S = 16.78 MB + X * 81.08 MB
```

3.1.D. Избыточность модели

Избыточность модели состоит из:

- Технического оверхеда arangodb журнала коллекции (4 MB)
- Технического оверхеда arangodb для каждой модели (shape id = 8 B, rev + key + id = 120 B)
- Избыточности самих данных:
    - Можно избавиться от всех ссылок и хранить документы в виде подмассивов и подобъектов
    - Можно избавиться от коллекции AstTrees

Получаем:

```
RawSize = CountUsers * 128
    + CountRepos * 128
    + CountBranches * 88
    + CountCommits * 232
    + CountCommitFiles * 200
    + CountAstNodes * 80
    + CountCommitBranches * 80

=> RawSize = 25.20 MB X

RawSize (X) / Size (X) = 0.311
```

Сделаем следующие выводы:

- Основная избыточность заключается в способе хранения ключей - они хранятся как строки
- Также довольно сильно увеличивают размер служебные атрибуты ArangoDB - _key, _id, _size
- В теории можно добиться сильного снижения размера БД на диске, но это повлечет за собой сильное увеличение сложности обращения к данным

3.1.E. Направление роста модели

Определим направление роста модели для каждой сущности как производную общего размера по количеству объектов данной сущности, приняв предыдущие константы

```
S'(CountUsers) = 81.08 MB
S'(CountRepos) = 16.25 MB
S'(CountBranches) = 3.24 MB
S'(CountCommits) = 162.15 KB
S'(CountCommitFiles) = 3.24 KB
S'(CountAstTrees) = 32.43 KB
S'(CountAstNodes) = 324 B
```

Как видим, рост линейный

3.1.F. Примеры данных

Модель

- Пользователь user
- Репозиторий repo
- Ветка master
- Коммит initial commit
- Три файла:
    - README.md
    - Папка src
        - Файл Hello.java
- AST-дерево для файла Hello.java

Представление модели в БД

users:

```
{
  "_id": "users/user1",
  "_key": "user1",
  "username": "user",
  "password_hash": "...",
  "email": "user@example.com",
  "created_at": 1678886400000
}
```

repositories:


```
{
  "_id": "repositories/repo1",
  "_key": "repo1",
  "name": "repo",
  "owner": "users/user1",
  "original_link": "git://github.com/user/project.git",
  "visibility": "public",
  "created_at": 1678886401000
}
```


branches:


```
{
  "_id": "branches/branch1",
  "_key": "branch1",
  "name": "master",
  "repo": "repositories/repo1",
  "created_at": 1678886402000
}
```


commits:


```
{
  "_id": "commits/commit1",
  "_key": "commit1",
  "branch": "branches/branch1",
  "hash": "abc123",
  "author": "user",
  "email": "user@example.com",
  "message": "initial commit",
  "files_changed": 3,
  "lines_added": 3,
  "lines_removed": 3,
  "created_at": 1678886403000,
  "root_files": [
    "commit_files/file1",
    "commit_files/file2"
  ]
}
```


commit_files:


```
{
  "_id": "commit_files/file1",
  "_key": "file1",
  "name": "README.md",
  "full_path": "README.md",
  "type": "file",
  "hash":
"2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824",
  "commit": "commits/commit1"
},
{
```

```
"_id": "commit_files/file2",
"_key": "file2",
"name": "src",
"full_path": "src",
"type": "directory",
"commit": "commits/commit1"
},
{
"_id": "commit_files/file3",
"_key": "file3",
"name": "Hello.java",
"full_path": "src/Hello.java",
"type": "file",
"hash":
"d5c5eab622d4e7daf297199549c9bacd54eb280b428234751c9ea49da0ee8e9e",
"commit": "commits/commit1",
"parent": "commit_files/file2"
}
```


branch_commits:


```
"_from": "branches/branch1",
"_to": "commits/commit1"
```


ast_parents:


```
"_from": "ast_nodes/ast_node1",
"_to": "ast_nodes/ast_node2"
```


ast_trees:


```
{
  "_id":
"ast_trees/d5c5eab622d4e7daf297199549c9bacd54eb280b428234751c9ea49da0ee
8e9e",
  "_key":
"d5c5eab622d4e7daf297199549c9bacd54eb280b428234751c9ea49da0ee8e9e",
  "_rev": "1",
  "created_at": 1234567894,
  "root_node": "ast_nodes/1"
}
```


ast_nodes:


```
[
  {
    "_id": "ast_nodes/1",
    "_key": "1",
    "_rev": "1",
    "label": "",
    "type": "ROOT"
  },
  {
    "_id": "ast_nodes/2",
```

```
    "_key": "2",
    "_rev": "2",
    "label": "Main",
    "type": "Class"
  },
]
```

3.1.G. Примеры запросов

Получение AST-дерева для файла в коммите

```
LET file = DOCUMENT("commit_files/key")
LET ast_tree = DOCUMENT(CONCAT("ast_trees/", file.hash))

RETURN ast_tree == null ? null : (
  FOR v IN OUTBOUND DOCUMENT(ast_tree.root_node) ast_parents
  RETURN v
)
```

Получение файлов в файловом дереве коммита, в которых есть вызов функции

```
LET targetCommit = DOCUMENT("commits/commit1")

LET commitFiles = (
  FOR file IN commit_files
    FILTER file.commit == targetCommit._id
    FILTER file.type == "file"
    RETURN file
)

LET filesWithFunction = (
  FOR file IN commitFiles
    LET astTree = DOCUMENT(CONCAT("ast_trees/", file.hash))
    FILTER astTree != null

    LET hasFunctionNode = FIRST(
      FOR v IN 0..10000 OUTBOUND DOCUMENT(astTree.root_node)
ast_parents
        FILTER v.type == "FunctionCall" AND v.label == "Main"
        LIMIT 1
        RETURN true
    )

    FILTER hasFunctionNode == true
    RETURN file
)

RETURN filesWithFunction
```

Получение всех коммитов для конкретного пользователя

```
LET targetUser = "user"
FOR c IN commits
    FILTER c.author == targetUser
    SORT c.created_at DESC
    RETURN c
```

Получение самых частых типов файлов, встречающихся в commit_files по всем коммитам

```
FOR cf IN commit_files
FILTER cf.type == 'file' AND CONTAINS(cf.name, ".")
LET parts = SPLIT(cf.name, ".")
FILTER LENGTH(parts) > 1
LET extension = LOWER(parts[-1])

COLLECT ext = extension WITH COUNT INTO occurrences

SORT occurrences DESC
LIMIT 10
RETURN {
file_extension: ext,
count_in_commit_files: occurrences
}
```

3.2. Реляционная модель

3.2.А. Графическое представление

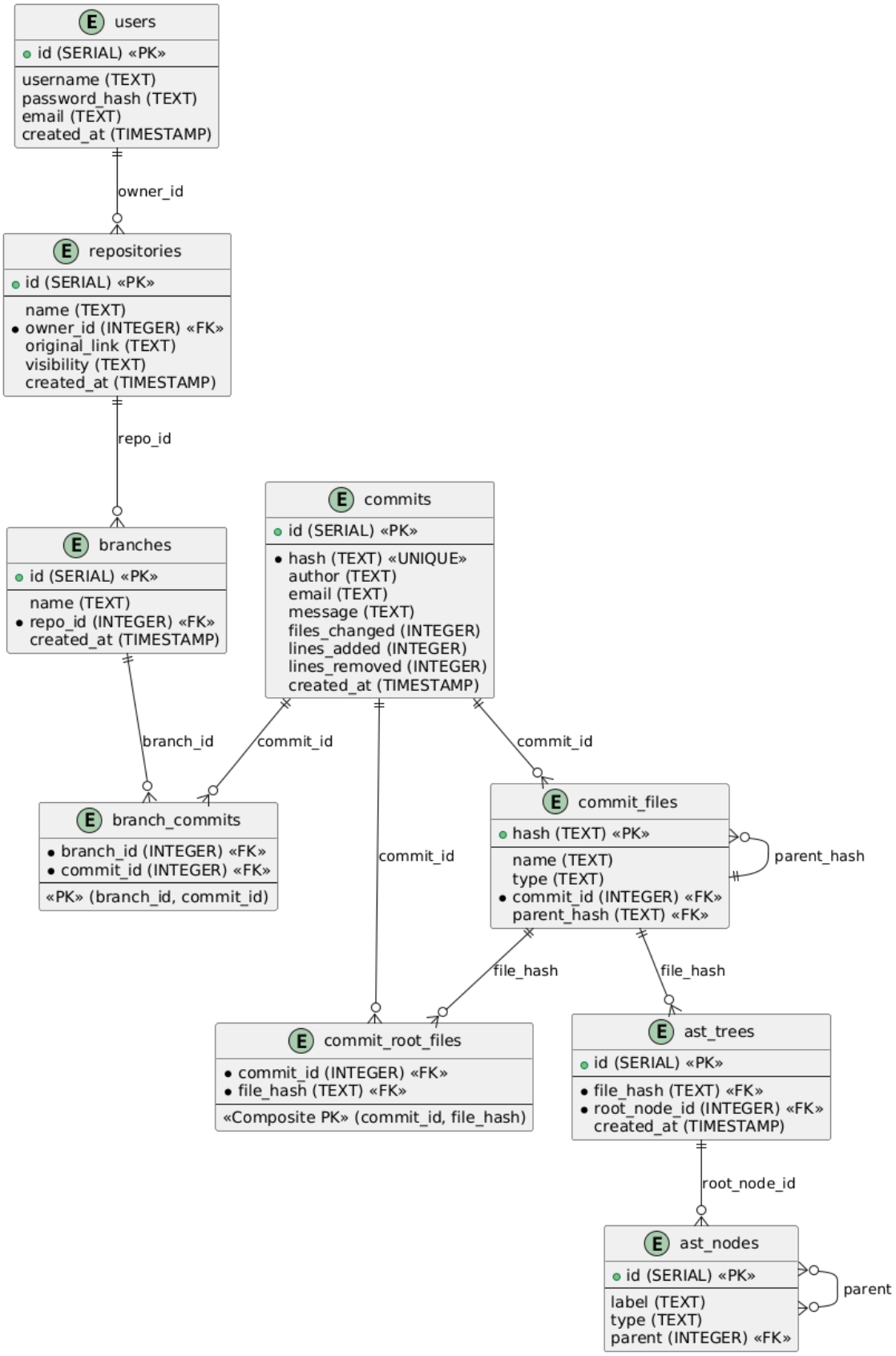

Рисунок 4 – графическое представление реляционной модели

3.2.B. Назначение коллекций, типы данных и сущностей

Размеры типов данных

- TEXT: 4 (length) + 4 * StrSize
- INTEGER: 4
- TIMSESTAMP: 8

users (хранение пользователей):

- id: SERIAL (PRIMARY KEY)
- username: TEXT
- password_hash: TEXT
- email: TEXT
- created_at: TIMESTAMP

repositories (хранение репозиториев):

- id: SERIAL (PRIMARY KEY)
- name: TEXT
- owner_id: INTEGER [-> users.id]
- original_link: TEXT
- visibility: TEXT
- created_at: TIMESTAMP

branches (хранение веток):

- id: SERIAL (PRIMARY KEY)
- name: TEXT
- repo_id: INTEGER [-> repositories.id]
- created_at: TIMESTAMP

commits (хранение коммитов):

- id: SERIAL (PRIMARY KEY)
- hash: TEXT <<UNIQUE>>
- author: TEXT
- email: TEXT
- message: TEXT
- files_changed: INTEGER
- lines_added: INTEGER
- lines_removed: INTEGER
- created_at: TIMESTAMP

branch_commits (связь веток и коммитов, многие-ко-многим):

- branch_id: INTEGER [-> branches.id]
- commit_id: INTEGER [-> commits.id]
- PRIMARY KEY (branch_id, commit_id)

commit_root_files (связь коммитов и корневых файлов):

- commit_id: INTEGER [-> commits.id]
- file_id: INTEGER [-> commit_files.id]
- PRIMARY KEY (commit_id, file_id)

commit_files (хранение файлов, дедуплицированных по хешу или уникальных в рамках коммита):

- id: SERIAL (PRIMARY KEY)
- hash: TEXT
- name: TEXT
- type: TEXT
- full_path: TEXT
- commit_id: INTEGER [-> commits.id]
- parent_id: INTEGER? [-> commit_files.id]

ast_trees (хранение AST-деревьев):

- id: SERIAL (PRIMARY KEY)
- file_hash: TEXT
- root_node_id: INTEGER [-> ast_nodes.id]
- created_at: TIMESTAMP

ast_nodes (хранение узлов AST):

- id: SERIAL (PRIMARY KEY)
- label: TEXT
- type: TEXT
- parent: INTEGER? [-> ast_nodes.id]

3.2.C. Оценка объема информации, хранимой в модели

Примем следующие константы:

- AvgStringLength: 9
- AvgRootFilesPerCommit: 20

Получим следующие формулы для размера моделей:

- users: 4 + 40 * 3 + 8 = 132
- repositories: 4 * 2 + 40 * 3 + 8 = 136
- branches: 4 * 2 + 40 + 8 = 56
- commits: 4 * 4 + 40 * 4 + 8 = 200
- branch_commits : 4 * 2 = 8
- commit_root_files: 4 * 2 = 8
- commit_files: 4 * 3 + 40 * 4 = 172
- ast_trees: 4 * 2 + 8 + 8 = 24
- ast_nodes: 4 * 2 + 40 * 2 = 88

Оценим общий размер БД:

```
CountRepos = CountUsers * ReposPerUser
CountBranches = CountRepos * BranchesPerRepo
CountCommits = CountBranches * CommitsPerBranch
CountCommitFiles = CountCommits * FilesPerCommit
CountCommitBranches = CountCommits * BranchesPerCommit
CountCommitRootFiles = 20 * CountCommits
# AST-деревья дедублицируются по ключу и не создаются для всех файлов в репозитории
CountAstTrees = CountRepos * UniqueSupportedFilesPerRepo
CountAstNodes = CountAstTrees * AstNodesPerTree

S = CountUsers * (132 + TableRowOverhead)
+ CountRepos * (136 + TableRowOverhead)
+ CountBranches * (56 + TableRowOverhead)
+ CountCommits * (200 + TableRowOverhead)
+ CountCommitFiles * (172 + TableRowOverhead)
+ CountCommitBranches * (8 + TableRowOverhead)
+ CountCommitRootFiles * (8 + TableRowOverhead)
+ CountAstTrees * (24 + TableRowOverhead)
+ CountAstNodes * (88 + TableRowOverhead)
+ TableOverhead * 9
```

Оценим зависимость размера от числа пользователей, приняв следующие константы:

```
CountUsers = X
ReposPerUser = 5
BranchesPerRepo = 5
CommitsPerBranch = 20
FilesPerCommit = 50
UniqueSupportedFilesPerRepo = BranchesPerRepo * CommitsPerBranch * FilesPerCommit / 10
TableOverhead = 8 * 1024 * 1024
TableRowOverhead = 32
AstNodesPerTree = 100
BranchesPerCommit = 2

=> S = 67.11 MB + X * 35.8 MB
```

3.2D. Избыточность модели

Избыточность модели состоит из:

- Технического оверхеда postgres для таблицы (8 MB)
- Технического оверхеда postgres для каждой строки (32 B)
- Избыточности самих данных:
    - Можно избавиться от всех ссылок и хранить данные в виде одного большого словаря

Получаем:

```
RawSize = CountUsers * 128
    + CountRepos * 128
    + CountBranches * 48
    + CountCommits * 196
    + CountCommitFiles * 160
    + CountAstNodes * 80

=> RawSize = 24.1 MB X

RawSize (X) / Size (X) = 0.67
```

Сделаем следующие выводы:

- Избытычность на каждую таблицу - больше, чем избыточность на каждую коллекцию
- Общий размер и избыточность для PostgreSQL значительно меньше, чем для ArangoDB

3.2.E. Направление роста модели

Определим направление роста модели для каждой сущности как производную общего размера по количеству объектов данной сущности, приняв предыдущие константы

```
S'(CountUsers) = 35.8 MB
S'(CountRepos) = 7.16 MB
S'(CountBranches) = 1.43 MB
S'(CountCommits) = 71.6 KB
S'(CountCommitFiles) = 1.43 KB
S'(CountAstTrees) = 14.32 KB
S'(CountAstNodes) = 143.19 B
```

Как видим, рост линейный

3.2.F. Примеры данных

Модель:

- Пользователь user
- Репозиторий repo
- Ветка master
- Коммит initial commit
- Три файла:
  - README.md
  - Папка src/
    - Файл Hello.java
- AST-дерево для файла Hello.java

Представление модели в БД:

```
INSERT INTO users (id, username, password_hash, email, created_at)
VALUES (1,
        'user',
        '...',
        'user@example.com',
        NOW());

INSERT INTO repositories (id, name, owner_id, original_link,
visibility, created_at)
VALUES (1,
        'repo',
        1,
        'git://github.com/user/project.git',
        'public',
        NOW());

INSERT INTO branches (id, name, repo_id, created_at)
VALUES (1,
        'master',
        1,
        NOW());

INSERT INTO commits (id, hash, author, email, message,
files_changed, lines_added, lines_removed, created_at)
VALUES (1,
        'abc123def456',
        'user',
        'user@example.com',
        'initial commit',
        3,
        10,
        0,
```

```
        NOW());

INSERT INTO branch_commits (branch_id, commit_id)
VALUES (1,
        1);

INSERT INTO commit_files (id, hash, name, type, commit_id,
parent_id, full_path)
VALUES (1,

'2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824'
,
        'README.md',
        'file',
        1,
        NULL,
        'README.md');

INSERT INTO commit_files (id, hash, name, type, commit_id,
parent_id, full_path)
VALUES (2,
        null,
        'src',
        'directory',
        1,
        NULL,
        'src');

INSERT INTO commit_files (id, hash, name, type, commit_id,
parent_id, full_path)
VALUES (3,

'd5c5eab622d4e7daf297199549c9bacd54eb280b428234751c9ea49da0ee8e9e'
,
        'Hello.java',
        'file',
        1,
        2,
        'src/Hello.java');

INSERT INTO commit_root_files (commit_id, file_id)
VALUES (1,
        1);

INSERT INTO commit_root_files (commit_id, file_id)
VALUES (1,
        2);

INSERT INTO ast_nodes (id, label, type, parent)
VALUES (1, '', 'ROOT', NULL),
       (2, 'Main', 'Class', 1);

INSERT INTO ast_trees (id, file_hash, root_node_id, created_at)
VALUES (1,

'd5c5eab622d4e7daf297199549c9bacd54eb280b428234751c9ea49da0ee8e9e'
,
```

```
        1,
        NOW());
```

3.2.G. Примеры запросов

Получение AST-дерева для файла в коммите:

```
WITH ast_tree AS (
  SELECT * FROM ast_trees WHERE file_hash = :commit_file_hash
)
SELECT
  an.id,
  an.label,
  an.type,
  an.parent
FROM ast_nodes an
WHERE an.id IN (
  WITH RECURSIVE ast_tree_nodes AS (
     SELECT id FROM ast_nodes WHERE id = (SELECT root_node_id FROM
ast_tree)
    UNION ALL
    SELECT an.id
    FROM ast_nodes an
           JOIN ast_tree_nodes atn ON an.parent = atn.id
  )
  SELECT id FROM ast_tree_nodes
);
```

Получение файлов в файловом дереве коммита, в которых есть вызов функции

```
WITH commit_files_filtered AS (
  SELECT cf.*
  FROM commit_files cf
  WHERE cf.commit_id = :commit_id
    AND cf.type = 'file'
),
     files_with_function AS (
       SELECT cf.*
       FROM commit_files_filtered cf
              JOIN ast_trees at ON at.file_hash = cf.hash
WHERE EXISTS (
  WITH RECURSIVE ast_tree_nodes AS (
  SELECT id, label, type, parent
  FROM ast_nodes
  WHERE id = at.root_node_id

  UNION ALL

  SELECT an.id, an.label, an.type, an.parent
  FROM ast_nodes an
  JOIN ast_tree_nodes atn ON an.parent = atn.id
  )
  SELECT 1
```

```
  FROM ast_tree_nodes
  WHERE type = 'FunctionCall' AND label = 'Main'
  LIMIT 1
  )
  )
SELECT * FROM files_with_function
```

Поиск самых глубоких (по количеству уровней) AST деревьев

```
WITH RECURSIVE node_depths AS (
  SELECT
    "at".id AS ast_tree_id,
    "at".file_hash,
    "at".root_node_id AS node_id,
    1 AS depth
  FROM ast_trees "at"
  UNION ALL
  SELECT
    nd.ast_tree_id,
    nd.file_hash,
    an.id AS node_id,
    nd.depth + 1 AS depth
  FROM ast_nodes an
         JOIN node_depths nd ON an.parent = nd.node_id
),
               max_depth_per_tree AS (
                 SELECT
                   ast_tree_id,
                   file_hash,
                   MAX(depth) AS max_tree_depth
                 FROM node_depths
                 GROUP BY ast_tree_id, file_hash
               )
SELECT
  mdpt.ast_tree_id,
  mdpt.file_hash,
  cf.name AS file_name,
  mdpt.max_tree_depth
FROM max_depth_per_tree mdpt
        LEFT JOIN commit_files cf ON mdpt.file_hash = cf.hash AND
cf.type = 'file'
ORDER BY mdpt.max_tree_depth DESC
  LIMIT 10;
```

Поиск коммита, который более всего изменил AST дерево

```
WITH RECURSIVE branch_commit_history AS (
  SELECT
    c.id AS commit_id,
    c.hash AS commit_hash_val,
    c.created_at,
    ROW_NUMBER() OVER (ORDER BY c.created_at ASC, c.id ASC) as rn
  FROM commits c
         JOIN branch_commits bc ON c.id = bc.commit_id
  WHERE bc.branch_id = :target_branch_id
),
```

```
                file_versions_in_branch AS (
                  SELECT
                    bch.commit_id,
                    bch.commit_hash_val,
                    bch.rn,
                    cf.hash AS file_hash_in_commit,
                    cf.id AS commit_file_id
                  FROM branch_commit_history bch
                           JOIN commit_files cf ON bch.commit_id =
cf.commit_id
                  WHERE cf.full_path = :target_full_path
                    AND cf.type = 'file' AND cf.hash IS NOT NULL
                ),
                ast_sizes_for_file_versions AS (
                  SELECT
                    fvib.commit_id,
                    fvib.commit_hash_val,
                    fvib.rn,
                    fvib.file_hash_in_commit,
                    (SELECT COUNT(*)
                     FROM (
                           WITH RECURSIVE ast_tree_nodes AS (
                               SELECT id FROM ast_nodes WHERE id =
"at".root_node_id
                             UNION ALL
                             SELECT child_an.id
                             FROM ast_nodes child_an
                                    JOIN ast_tree_nodes parent_atn
ON child_an.parent = parent_atn.id
                           )
                           SELECT id FROM ast_tree_nodes
                         ) nodes_in_tree
                    ) AS ast_node_count
                  FROM file_versions_in_branch fvib
                           JOIN ast_trees "at" ON "at".file_hash =
fvib.file_hash_in_commit
                ),
                ast_size_changes AS (
                  SELECT
                    current_ast.commit_id,
                    current_ast.commit_hash_val,
                    current_ast.rn,
                    current_ast.file_hash_in_commit,
                    current_ast.ast_node_count,
                        LAG(current_ast.ast_node_count, 1, 0) OVER
(ORDER BY current_ast.rn ASC) AS prev_ast_node_count,
                                ABS(current_ast.ast_node_count -
LAG(current_ast.ast_node_count,    1,    0)    OVER    (ORDER    BY
current_ast.rn ASC)) AS ast_size_diff
                  FROM ast_sizes_for_file_versions current_ast
                )
SELECT
  asc_change.commit_id,
  asc_change.commit_hash_val AS commit_hash,
  c.message AS commit_message,
  asc_change.file_hash_in_commit,
  asc_change.prev_ast_node_count,
```

```
  asc_change.ast_node_count,
  asc_change.ast_size_diff
FROM ast_size_changes asc_change
       JOIN commits c ON asc_change.commit_id = c.id
WHERE asc_change.rn > 1
ORDER BY asc_change.ast_size_diff DESC
  LIMIT 1;
```

3.3. Сравнение моделей

3.3.A. Объём информации

Таблица 1 – Объём информации

Модель	Базовый размер	На 1 пользователя	Формула роста
Нереляционная (arango)	16.78 MB	81.08 MB	16.78 MB + X * 81.08 MB
Реляционная (postgres)	67.11 MB	35.8 MB	67.11 MB + X * 35.8 MB

3.3.B. Эффективность хранения

Таблица 2 – Эффективность хранения

Модель	Технический оверхед	RawSize (X) / Size (X)
Нереляционная (arango)	4MB+128B	0.311
Реляционная (postgres)	8MB+32B	0.67

3.3.C. Масштабируемость

Таблица 3 – Масштабируемость

Сущность	Нереляционная (arango)	Реляционная (postgres)
Пользователь	81.08 MB	35.8 MB
Репозиторий	16.25 MB	7.16 MB
Ветка	3.24 MB	1.43 MB
Коммит	162.15 KB	71.6 KB

Файл	3.24 KB	1.43 KB
Узел AST	324 B	143 B

Основные запросы аналогичны по количеству обращений к БД и количеству использованных основных коллекций/таблиц.

3.3.D. Вывод

Современные реляционные БД имеют довольно широкий спектр применения; в данном случае видно, что использование реляционной БД на примере PostgreSQL превосходно подошло бы для проекта, особенно учитывая "взрослость" технологии и преимущества в эффективности хранения информации.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Краткое описание

Разработанное приложение представляет собой веб-систему для хостинга Git-репозиториев и анализа их исходного кода, развертываемую с помощью Docker comopose и состоящую из трёх основных контейнеров: базы данных, серверной (backend) и клиентской (frontend) частей.

1. Контейнер базы данных: Обеспечивает хранение всей информации о пользователях, репозиториях, их файловой структуре, коммитах, ветках и сгенерированных AST-деревьях. Используется мультимодельная СУБД ArangoDB.

2. Backend: Реализован на Java с использованием фреймворка Spring Boot и библиотеки JGit для взаимодействия с Git-репозиториями. Backend запускается в контейнере на базе соответствующего образа с Java Runtime Environment. При запуске контейнера происходит инициализация необходимых коллекций документов и рёбер, что позволяет системе сразу приступить к работе. Все необходимые зависимости управляются через систему сборки Gradle и включаются в артефакт приложения, который копируется в рабочую директорию контейнера. Запуск осуществляется стандартными средствами Spring Boot. Backend предоставляет REST API для взаимодействия с системой, включая:

- Аутентификацию и авторизацию пользователей.
- Загрузку Git-репозиториев по URL с использованием JGit.
- Парсинг исходного кода для построения AST-деревьев.
- Предоставление данных о репозиториях, файлах, коммитах, ветках и AST-деревьях.
- Административные функции (управление пользователями, репозиториями, импорт/экспорт БД).
- Для административных задач, таких как импорт/экспорт всей базы данных, предусмотрены соответствующие эндпоинты.

3. Frontend: Это многостраничное веб-приложение, разработанное с использованием TypeScript, фреймворка React и экосистемы сопутствующих библиотек:

- Jotai для управления состоянием.
- TanStack Query для управления серверным состоянием, кэширования и синхронизации данных.
- TanStack Table для создания интерактивных таблиц.
- Shadcn/ui для построения пользовательского интерфейса.
- Tailwind CSS для стилизации.
- Vite в качестве сборщика и сервера для разработки.

Frontend разворачивается отдельно. Для разработки используется среда основной ОС компьютера разработчика, где устанавливаются зависимости из package.json и запускается dev-сервер Vite. Для production-сборки разработан Dockerfile, где выполняется команда bun run build, и полученные статические ассеты раздаются через легковесный веб-сервер Caddy в отдельном Docker-контейнере. Клиентская часть взаимодействует с backend через REST API, отображая информацию о репозиториях, файлах и AST-деревьях в удобном и наглядном виде. Пользователь может регистрироваться, загружать репозитории, просматривать их содержимое и AST-деревья, переключаться между ветками и коммитами. Администраторы имеют доступ к панели управления пользователями и всеми репозиториями с возможностями фильтрации, сортировки и просмотра детальной информации, реализованными с помощью TanStack Table.

Вся система полностью контейнеризирована, что обеспечивает простоту развертывания, масштабируемость и независимость от среды пользователя. Это также упрощает обновление отдельных компонентов системы, например, парсеров для новых языков программирования или будущих аналитических модулей.

Реализована фильтрация, поиск, сортировка всех сущностей. Таблицы с сущностями созданы с пагинацией. AST-деревья и просмотр файлов визуализированы с помощью Apache Echarts.

Присутствует функционал продвинутого анализа аст деревьев (поиск декларации и использований классов).

4.2. Использованные технологии

БД: arangodb:3.12.4

backend: java + spring + jgit + spoon

frontend: react + jotai + tanstack query + tanstack table + shadcn + vite + tailwind + typescript.

DevOPS: Docker

4.3. Снимки экрана приложения

Экраны приложения и переходы между ними отображены на рисунках 5-9.

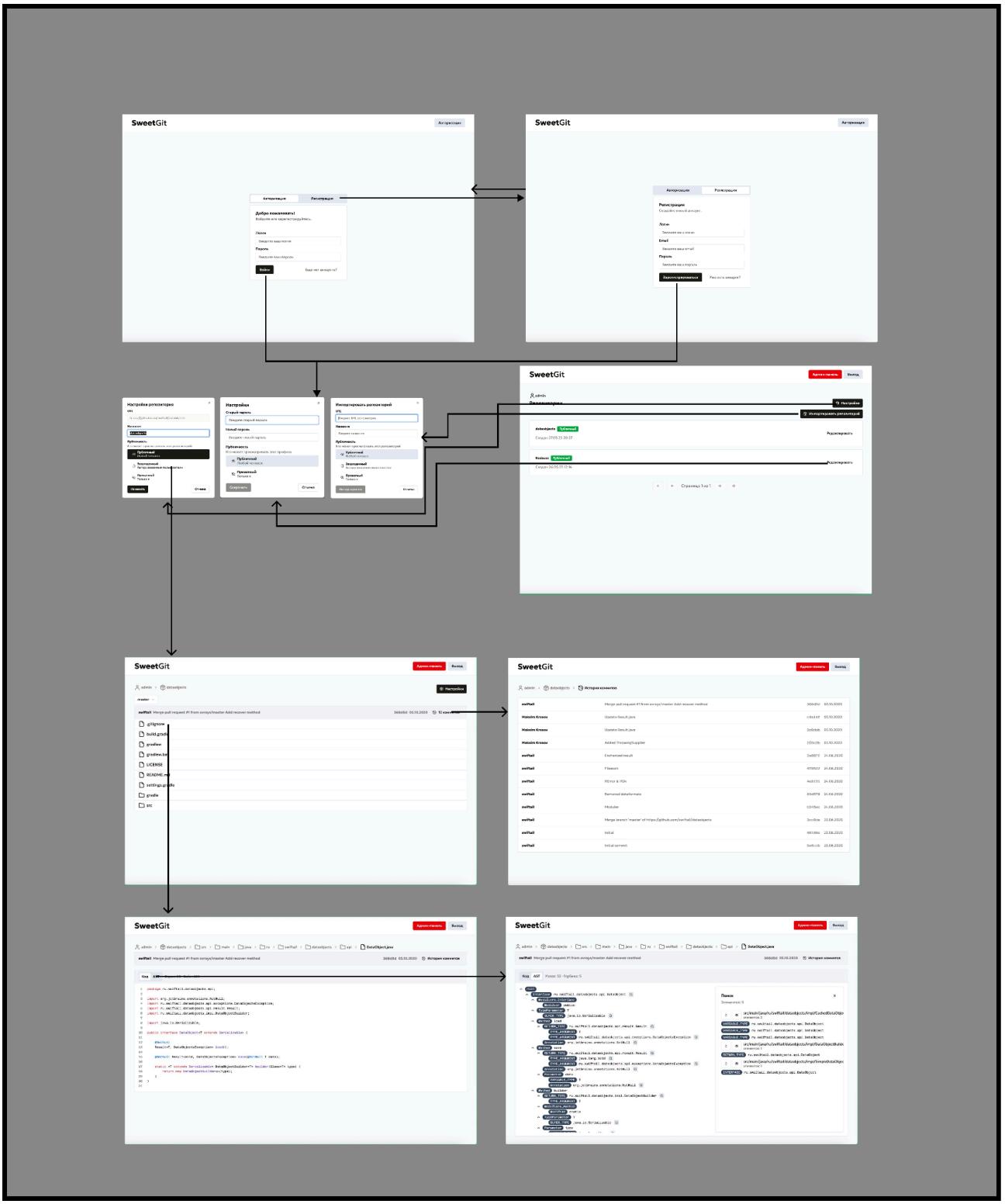

Рисунок 5. Схема экранов приложения.

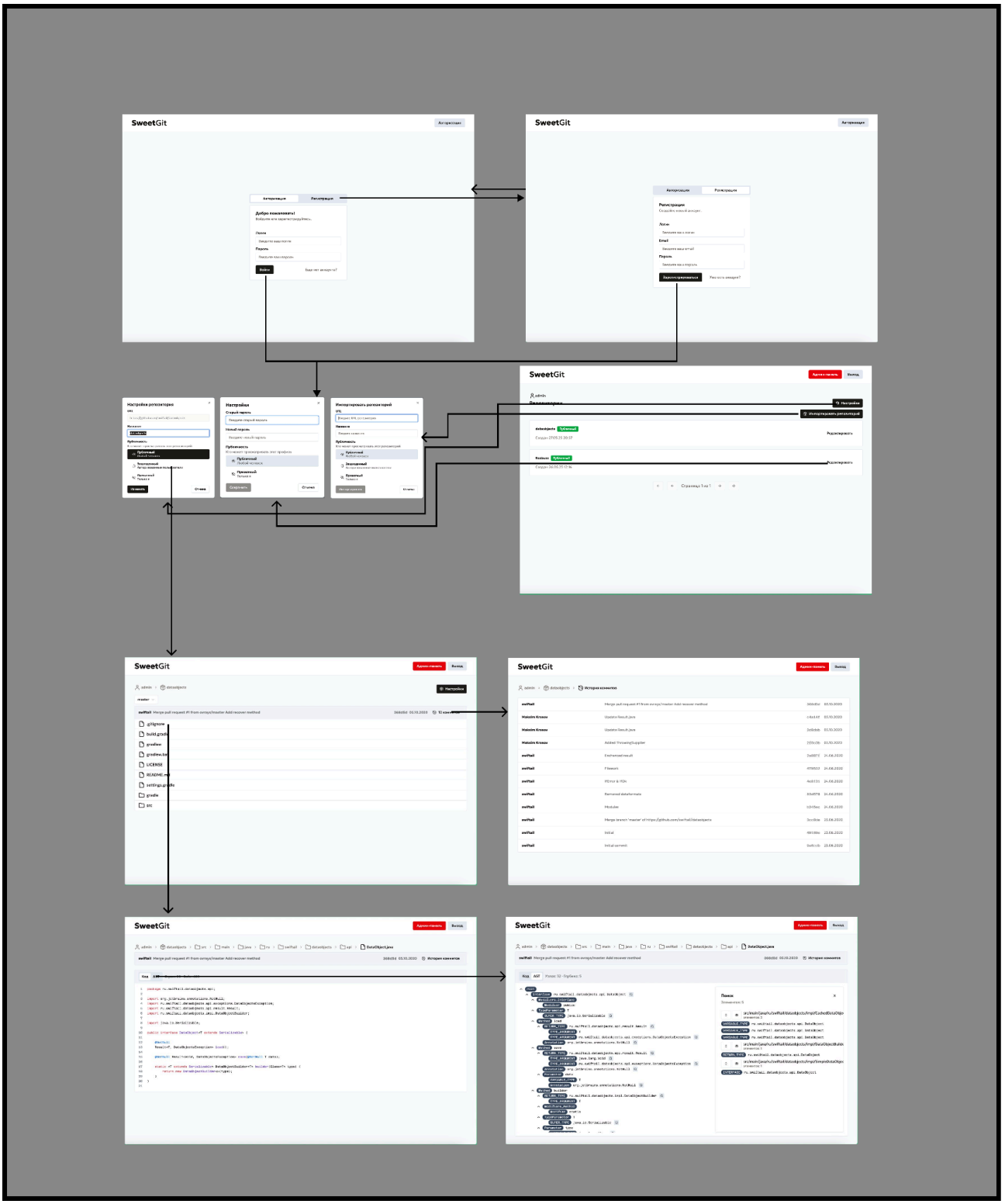

Рисунок 6. Схема экранов приложения.

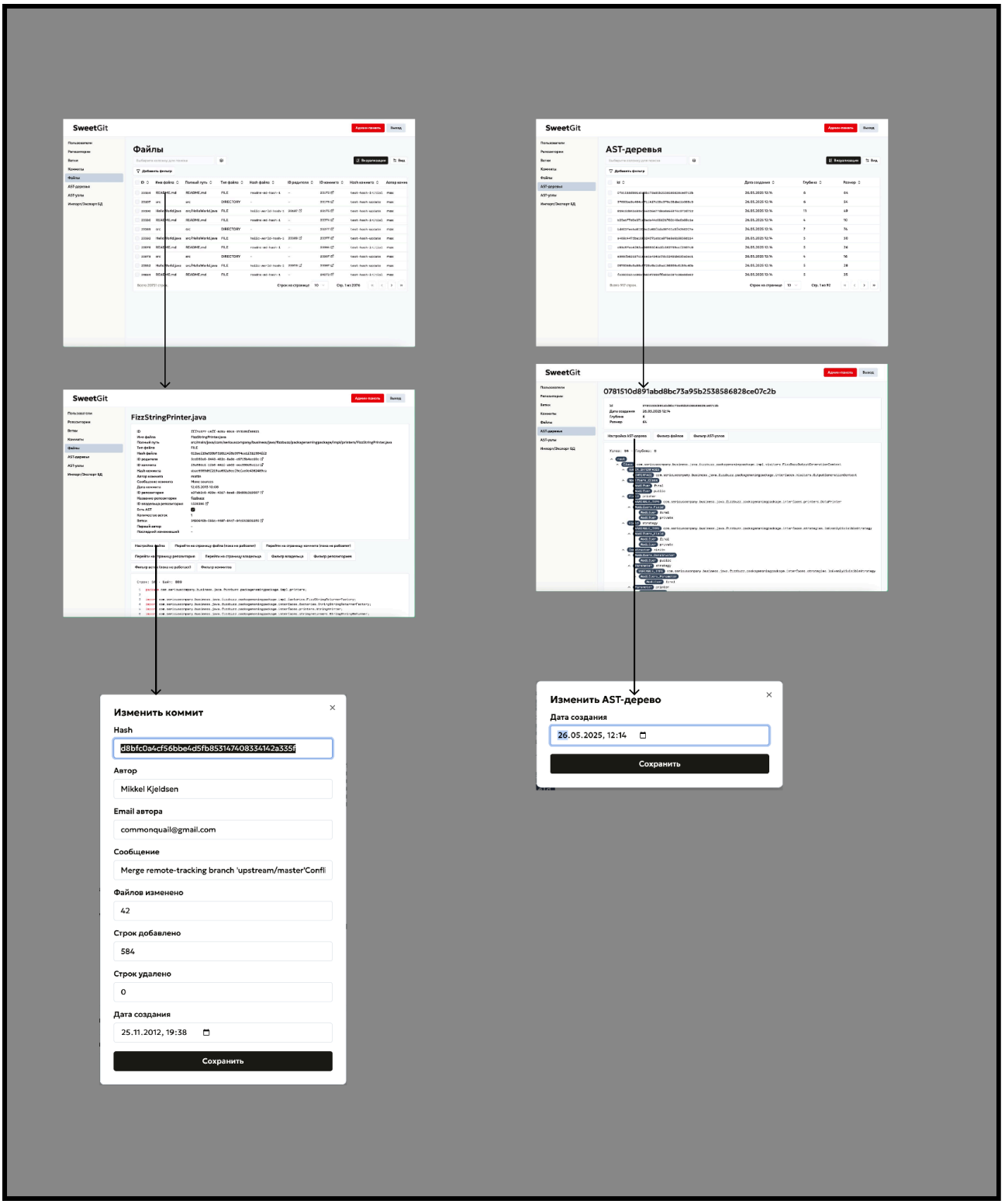

Рисунок 7. Схема экранов приложения.

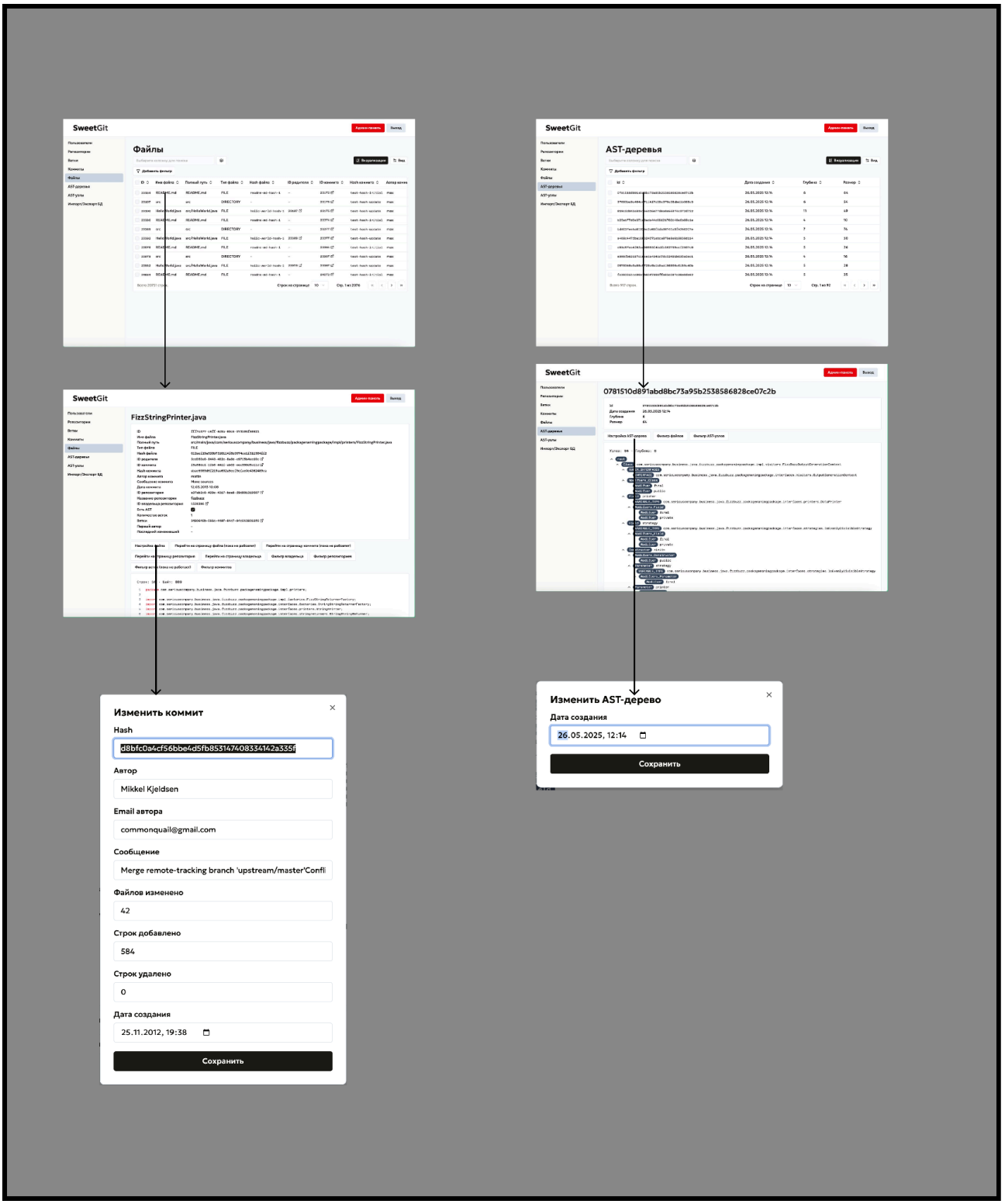

Рисунок 8. Схема экранов приложения.

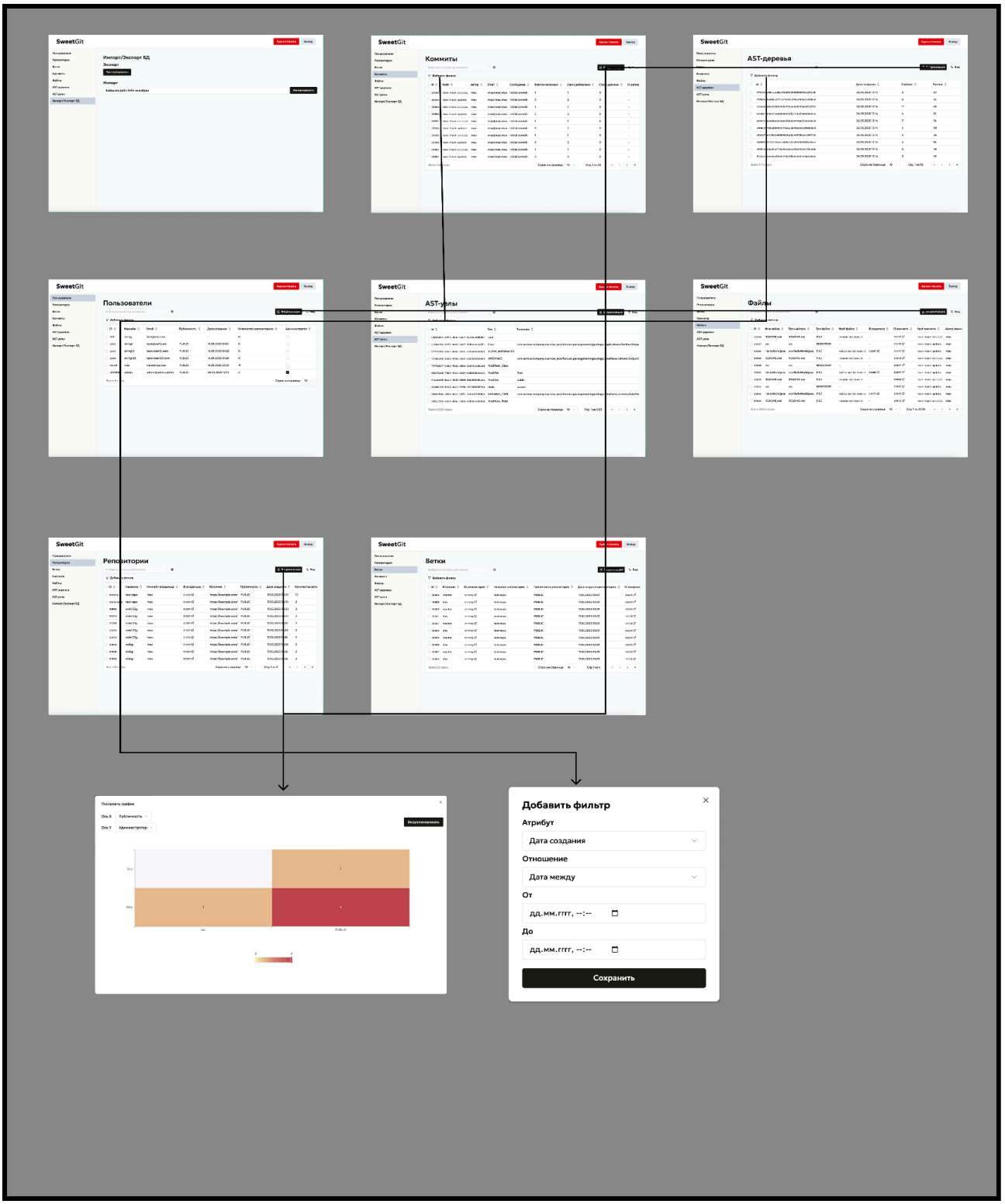

Рисунок 9. Схема экранов приложения.

Ссылка на приложение

Ссылка на github-репозиторий: https://github.com/moevm/nosql1h25-asttrees

ЗАКЛЮЧЕНИЕ

5.1. Достигнутые результаты

В результате выполнения работы был разработан и реализован сервис для хостинга Git-репозиториев с функциями углубленного анализа исходного кода, основанный на мультимодельной (документно-графовой) модели данных в ArangoDB. В процессе работы была создана подробная схема базы данных (представленная ранее), иллюстрирующая коллекции документов и рёбер, используемые для хранения информации о пользователях, репозиториях, их версиях, файлах и детальных AST-представлениях. Для удобства развертывания, тестирования и эксплуатации всей системы использованы современные инструменты контейнеризации, такие как Docker и docker-compose.

Фронтенд-часть приложения, построенная на React, TypeScript и сопутствующих библиотеках (Jotai, TanStack Query/Table, Shadcn/ui, Tailwind CSS), обеспечивает многофункциональный и интуитивно понятный пользовательский интерфейс:

- Пользователи могут регистрироваться, авторизовываться, загружать Git-репозитории по URL.
- Осуществляется просмотр файловой структуры репозиториев, содержимого файлов и их AST-деревьев на различных ветках и коммитах.
- Доступны результаты продвинутого анализа AST-деревьев, такие как поиск деклараций и использований классов/функций/переменных.
- Администраторам доступна полнофункциональная панель управления пользователями и всеми репозиториями в системе, включая возможности для импорта/экспорта данных и визуализации статистики (с использованием Apache ECharts).

Бэкенд, реализованный на Java с использованием Spring Boot и JGit, инкапсулирует всю основную бизнес-логику и аналитические возможности:

- Обрабатывает запросы от клиента, выполняет валидацию данных и обеспечивает безопасное и эффективное взаимодействие с базой данных ArangoDB.
- Осуществляет управление Git-операциями (клонирование, получение информации о ветках и коммитах).
- Реализует парсинг исходного кода для и построение AST-деревьев.
- Проводит серверный анализ AST-деревьев, включая поиск деклараций и использований сущностей кода.
- Предоставляет мощные серверные механизмы фильтрации (по множеству критериев), полнотекстового поиска, гибкой сортировки и пагинации по всем ключевым данным (пользователи, репозитории, коммиты, файлы), что обеспечивает высокую производительность даже при больших объемах информации.
- Готовит агрегированные данные для визуализации статистики на стороне клиента.

Взаимодействие между фронтендом и бэкендом осуществляется через REST API. Приложение поддерживает основные CRUD-операции для управления пользователями и репозиториями, а также реализует сложный функционал, связанный с обработкой и анализом исходного кода.

5.2. Недостатки и пути для улучшения полученного решения

В процессе разработки и эксплуатации сервиса были выявлены определённые аспекты, требующие доработки для повышения удобства использования и расширения функциональных возможностей:

1. Визуализация связанных сущностей: В некоторых разделах интерфейса при отображении связанных данных (например, при просмотре истории коммитов или связей в AST-деревьях) выводятся внутренние

идентификаторы вместо более информативных наименований или ключевых атрибутов сущностей. Это может затруднять быстрое восприятие информации.

2. Функционал просмотра репозитория: Хотя реализован базовый просмотр файлов и структуры репозитория, включая навигацию по веткам и коммитам, а также просмотр AST, интерфейс и набор инструментов пока уступают по полноте и интуитивности ведущим платформам хостинга кода (например, GitHub).
3. Адаптивная верстка: Текущая реализация пользовательского интерфейса не в полной мере оптимизирована для работы на устройствах с различными разрешениями экрана (например, на мобильных устройствах или планшетах).
4. Обработка ошибок и валидация: Несмотря на реализованные механизмы, существует потенциал для более детальной и информативной обработки ошибок на стороне клиента и сервера, а также для расширения валидации вводимых пользователем данных для повышения общей надежности системы.

5.3. Будущее развитие решения

Для дальнейшего повышения ценности и конкурентоспособности сервиса возможны следующие направления развития:

1. Расширение поддержки языков программирования: На текущий момент система поддерживает парсинг и анализ AST для ограниченного набора языков (только для java).
2. Углубление аналитических возможностей AST: Существующий функционал анализа AST (например, поиск деклараций и использований) является хорошей основой, но возможна разработка и внедрение новых, более сложных алгоритмов анализа AST для выявления широкого спектра паттернов кода, потенциальных уязвимостей, оценки сложности, метрик качества или предсказания проблемных зон.

3. Реализация полноценной функциональности Git-сервера: Текущая модель работы предполагает однократный импорт репозитория для анализа. Активное взаимодействие с репозиторием (push, pull) после импорта не предусмотрено.
4. Дальнейшее улучшение пользовательского интерфейса (UI/UX): Помимо устранения текущих недостатков, есть потенциал для внедрения новых элементов управления, более продвинутых визуализаций и улучшения общего опыта взаимодействия.

ПРИЛОЖЕНИЯ

6.1. Документация по сборке и развертыванию приложения

Для сборки и развертывания приложения используется система контейнеризации Docker и инструмент docker-compose, что обеспечивает простоту процесса запуска. В корне проекта расположен файл docker-compose.yml, в котором описаны необходимые сервисы: база данных и серверная часть приложения, а также клиентская. Для запуска приложения необходимо установить Docker и docker-compose, после чего выполнить команду: docker-compose up --build. После успешного запуска все необходимые сервисы будут автоматически подняты в отдельных контейнерах, а приложение станет доступно по указанному в конфигурации адресу. Все параметры подключения и переменные окружения настраиваются в файле .env.

6.2. Инструкция для пользователя

1. Установка необходимых программ

Перед началом работы убедитесь, что на вашем компьютере установлены:

- Docker Desktop (для Windows/macOS) или Docker Engine (для Linux).
- Git (для клонирования репозитория).

2. Клонирование репозитория

Откройте терминал (или командную строку) и выполните команду:

```
git clone https://github.com/moevm/nosql1h25-asttrees.git
```

Перейдите в папку проекта:

```
cd nosql1h25-asttrees
```

3. Сборка и запуск приложения

В терминале, находясь в корневой папке проекта, выполните команду:

```
docker-compose up --build
```

Дождитесь завершения сборки всех контейнеров (база данных ArangoDB, backend на Java/Spring, frontend на React) и их запуска. Этот процесс может занять некоторое время при первом запуске, так как Docker будет загружать необходимые образы и собирать ваши сервисы.

4. **Доступ к приложению**

После успешного запуска всех сервисов приложение будет доступно в вашем веб-браузере по адресу: http://localhost:80

5. **Работа с приложением**

В пользовательском интерфейсе вы сможете:

- Зарегистрироваться и авторизоваться в системе.
- Загружать (импортировать) Git-репозитории по URL-ссылке.
- Просматривать список своих репозиториев и их содержимое (файлы, папки).
- Переключаться между ветками и коммитами для просмотра состояния репозитория в разные моменты времени.
- Просматривать исходный код файлов и сгенерированные для них Абстрактные Синтаксические Деревья (AST).
- Использовать функции продвинутого анализа AST (например, поиск деклараций и использований сущностей кода).
- Администраторы имеют доступ к панели управления, где могут:
    - Просматривать всех пользователей и все репозитории в системе.
    - Использовать фильтры, поиск и сортировку по различным параметрам.
    - Просматривать статистические данные и визуализации.
    - Осуществлять импорт и экспорт базы данных.

6. **Остановка приложения**

Для остановки работы приложения нажмите Ctrl+C в терминале, где был запущен docker-compose up.
Либо, из другой терминальной сессии, находясь в той же папке проекта, выполните команду:

```
docker-compose down
```

Если вы хотите полностью удалить все контейнеры, сети и связанные с ними тома, включая данные в ArangoDB, используйте команду:

```
docker-compose down -v
```

Эта команда остановит все сервисы и удалит все созданные для проекта тома, что позволит начать работу с "чистого листа" при следующем запуске.

Если приложение не запускается, проверьте, что все сервисы корректно описаны в docker-compose.yml и что все необходимые файлы присутствуют.
Убедитесь, что порты, указанные в .env и docker-compose.yml, свободны и не заняты другими программами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация ArangoDB: https://www.arangodb.com/docs/stable/ - Основной ресурс по работе с СУБД ArangoDB, включая язык запросов AQL, графовые операции, драйверы и администрирование.
2. Документация Spring Boot: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ - Официальное руководство по фреймворку Spring Boot, используемому для разработки backend на Java.
3. React - Официальная документация: https://react.dev/learn - Руководства, основные концепции и справочник по библиотеке React для создания пользовательских интерфейсов.
4. TypeScript - Справочник: https://www.typescriptlang.org/docs/handbook/intro.html - Официальный справочник и руководства по языку TypeScript, используемому для разработки frontend-части.
5. TanStack Query - Документация: https://tanstack.com/query/latest/docs/react/overview - Руководство по мощной библиотеке для управления серверным состоянием в React-приложениях.
6. Документация Docker: https://docs.docker.com/ - Официальная документация по Docker и Docker Compose для контейнеризации и развертывания приложения.