ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО
ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Факультет Санкт-Петербургская школа физико-математических и компьютерных наук

Анисимова Карина Витальевна

Разработка приложения для резервного копирования и восстановления информации об организациях с веб-сервиса GitHub

Выпускная квалификационная работа - БАКАЛАВРСКАЯ РАБОТА
по направлению подготовки
01.03.02 Прикладная математика и информатика
образовательная программа «Прикладная математика и информатика»

Рецензент:
Иванов Дмитрий Аркадьевич
Huawei, Russia research Institute,
директор лаборатории
инновационных средств разработки

Руководитель:
Мухин Михаил Сергеевич
кандидат физико-математических
наук,
доцент, департамент информатики

Санкт-Петербург
2023

Содержание

Аннотация

GitHub-организации предоставляют централизованную платформу для координации работы множества пользователей. Однако возникающий риск потери данных или их повреждения внутри GitHub-организации вызывает серьезные опасения.

В данной работе предлагается разработать инструмент для создания резервных копий и восстановления GitHub-организаций, который позволит пользователям обеспечить безопасность своей интеллектуальной собственности и восстанавливать состояние организации в любой момент. В ходе обзора литературы выделены ключевые особенности GitHub. Исследование также определяет ограничения инструмента по типам метаданных и предлагает будущие улучшения, чтобы поддерживать другие типы метаданных.

Разработка данного инструмента имеет важное профессиональное значение для пользователей сервиса GitHub, поскольку он позволяет предотвращать потерю данных и их повреждение, а также поддерживать надежность и доступность информации.

Ключевые слова: GitHub организация, сохранность данных, резервное копирование, автоматическое восстановление

Abstract

This study discusses the importance of effective teamwork for successful software development projects and how GitHub organizations provide a centralized platform for coordinating the work of multiple users. However, the risk of data loss or corruption within a GitHub organization is a major concern, and regular migration to another service is not always a practical solution. The work proposes a tool for backing up and restoring a GitHub organization, which will allow users to ensure the safety of their intellectual property and restore the state of their organization at any time. The literature review highlights the key features of GitHub, including its support for team development practices, GitHub Actions, collaboration tools, and API. The study also delimits the scope of the tool to essential types of metadata and proposes future enhancements to support other types of metadata. The development of this tool is of professional significance for GitHub service users, enabling them to prevent data loss and corruption and maintain the reliability and accessibility of their data.

Введение

Введение в область

В современном мире разработка программного обеспечения стала ключевым аспектом бизнеса, эффективная командная работа играет важную роль в успехе проекта. Разработка программного обеспечения - это сложный процесс, который может занять много времени и ресурсов. Важно иметь эффективную команду, которая будет работать вместе для достижения общих целей. Однако координация усилий многих разработчиков может оказаться сложной задачей, особенно когда они работают удаленно из разных частей мира. В этом случае, GitHub может стать незаменимым инструментом для организации командной работы.

Организации GitHub предоставляют централизованную платформу для объединения людей и проектов в среде для совместной работы. Будь то проект с открытым исходным кодом или коммерческое предприятие, GitHub организации предоставляют основу для координации работы нескольких пользователей над одним и тем же проектом.

Одним из преимуществ GitHub является возможность контроля версий кода. Это означает, что каждый разработчик может работать над своей копией проекта, не боясь повредить код других членов команды. При необходимости, разработчики могут объединить свои изменения в один общий проект.

Организации GitHub являются не только платформой для хранения кода и управления проектами, но также позволяют командам разрабатывать программное обеспечение более эффективно. Один из способов улучшить командную работу на GitHub - это использовать функцию pull request. Это позволяет участникам организации создавать свои собственные ветки кода, вносить изменения и пред-

лагать их для включения в основную ветку. Это дает возможность другим членам команды просмотреть, обсудить и комментировать предложенные изменения перед их объединением с основной веткой. Также в GitHub есть возможность просматривать историю изменений и версий кода, что позволяет командам легко отслеживать, какие изменения были внесены и кем.

Еще одна важная функция GitHub - это система управления задачами, которая помогает командам управлять проектами, определять и назначать задачи и отслеживать их выполнение. Эта функция позволяет всей команде иметь общее понимание того, что нужно сделать, кто за что отвечает и какой прогресс уже достигнут. Все это делает командную работу более организованной и эффективной.

Приватные репозитории, доступные только членам команды, являются еще одним важным преимуществом GitHub. Они обеспечивают безопасное хранение конфиденциальных данных и позволяют ограничить доступ к коду только необходимым людям.

GitHub также предоставляет широкий выбор интеграций с инструментами CI/CD, такими как Jenkins, Travis CI и другими. Это позволяет автоматизировать процесс сборки, тестирования и развертывания проектов. Например, можно автоматически запускать тесты после каждого коммита в репозиторий и автоматически развертывать изменения на серверах. Такой подход упрощает процесс разработки и повышает качество кода, а также позволяет сократить время, необходимое для выпуска новых версий продукта.

В целом, использование организаций GitHub может значительно упростить процесс совместной работы команды и улучшить качество разработки программного обеспечения.

GitHub организация хранит огромные объемы данных, но эти данные подвержены различным рискам. В частности, сотрудники организации могут нечаянно или намеренно нанести вред репозиториям и компании в целом. Примером такого нанесения вреда может

быть изменение данных репозитория без возможности их восстановления или даже удаление целого репозитория. Такой сценарий потери доступа к данным, включая кодовую базу и все метаданные, может нанести серьезный ущерб организации.

В случае возникновения сбоев или ошибок в работе сервиса GitHub, возможна потеря доступа к данным организации, что может негативно сказаться на ее работе и производительности. Более того, ошибки и сбои в работе сервиса могут привести к утечке данных и другим серьезным последствиям для организации.

Более того, в последнее время серьезной проблемой стал риск санкций и последующей блокировки компаний. В таких ситуациях влияние блокировки на организацию может быть катастрофическим, так как может привести к потере доступа ко всем данным без возможности восстановления. Этот сценарий может привести к серьезным последствиям для компании, включая потерю доходов, ущерб репутации и сбои в работе.

Постановка цели и задачи

Когда дело доходит до управления данными и их хранения, очень важно иметь надежную и эффективную систему для предотвращения любой потенциальной потери или повреждения важной информации. Универсальное решение — использовать инструменты для резервного копирования. Регулярно выполняя резервное копирование данных, пользователи могут быть уверены, что у них всегда будет надежная и актуальная копия организации GitHub, которую можно легко восстановить на GitHub в случае потери или повреждения данных, однако GitHub таких возможностей не предоставляет.

В связи с этим основной целью данной работы является разработка инструмента для резервного копирования и восстановления организации GitHub. Для достижения поставленной цели были вы-

делены следующие задачи:

1. Выбор оптимального подхода и архитектуры
2. Реализация резервного копирования git-данных и метаданных
3. Реализация восстановления git-данных и метаданных
4. Тестирование
5. Сравнение с аналогами

Ограничения

Организация GitHub содержит различные типы метаданных, не все из которых считаются очень важными. Поэтому было принято решение поддерживать резервное копирование основных типов метаданных: участников, issues и pull requests. При разработке пайплайна необходимо было обеспечить возможность добавления резервного копирования для других типов метаданных максимально простым способом.

Значимость

Успешная разработка этого инструмента позволит пользователям сервиса GitHub не беспокоиться о сохранности своей интеллектуальной собственности. Используя результаты этой работы, пользователи смогут в любой момент восстановить состояние своей организации.

Структура работы

В главе 1 представлен обзор существующих полных или частичных решений существующей проблемы.

В главе 2 представлено описание предлагаемого решения, представлены технические детали итоговой реализации.

В главе 3 представлено описание процесса тестирования полученного решения, также представлены результаты апробации решения на реальных GitHub организациях.

В главе 4 приводится описание процесса сравнения полученного решения с возможными аналогами, представлены выводы о качестве работы итоговой реализации.

1 Обзор аналогов

1.1 Миграция

Существует несколько методов для обеспечения сохранности данных организации в GitHub. Один из них заключается в регулярном переносе данных на другой сервис - миграции.

Например, популярный GitHub - подобный сервис GitLab [11] позволяет импортировать репозиторий [16] с GitHub. Данная функция включает в себя как импорт содержимого репозиториев, так и все метаданные. Используя данную возможность можно настроить регулярный перенос всех репозиториев. Однако, следует учитывать, что данная опция не решает полностью проблему сохранности данных организации. Например, данные об участниках и другие специфические для организации метаданные будут утрачены в процессе импорта на другую платформу.

Также такую возможность предоставляет сервис Gitea [7]. Однако у такого подхода есть существенный недостаток - восстановление организации обратно на GitHub не является простой задачей. Сам сервис GitHub не предоставляет такой функциональности, но существует ряд инструментов, которые могут пригодиться в такой ситуации, хотя они далеки от идеальных.

Например, инструмент Gitlab-github-migrate [13] позволяет перемещать репозитории с GitLab обратно на GitHub. Однако данный инструмент работает только на уровне отдельных репозиториев, а не на уровне всей организации. Кроме того, возникают проблемы с восстановлением некоторых метаданных репозиториев, например, все pull requests никак не переносятся.

Ещё одним инструментом для восстановления организации на GitHub является Node-gitlab-2-github [19]. Однако, он также ограничен использованием на уровне отдельных репозиториев, а не на уровне всей организации. В отличие от Gitlab-github-migrate, этот инструмент крайне сложен в использовании, так как требует значительного количества предварительной подготовки данных, например, соответствия логинов пользователей между GitHub и GitLab, а также правильной настройки переноса. Кроме того, имеются проблемы с переносом метаданных, таких как review, а также часто возникают проблемы с переносом issues.

Проблема восстановления на GitHub с других сервисов является особенно проблематичной, и в этом контексте инструмент для перемещения репозитория с Gitea - Gitea-github-sync [8] представляется как один из редких возможных вариантов. Однако, необходимо отметить, что данный инструмент, переносит только содержимое репозиториев и не включает в себя метаданные, что, по большей части, может оказаться бесполезным.

В результате, подход использования регулярного переноса организации на иной сервис может быть полезен для защиты от возможных угроз безопасности и потери доступа к данным в GitHub. Однако, перенос на другой сервис не является безопасным и идеальным решением, так как при переносе обратно на GitHub могут возникнуть трудности, связанные с отсутствием необходимых инструментов. Кроме того, существующие способы обратного переноса не идеальны, при переносе данных теряется значительная часть важных

метаданных, что приводит к потере ценных данных и затрудняет их последующую обработку.

1.2 Зеркалирование

Ещё один вариант решения проблемы сохранности данных в GitHub организации - это использование зеркалирования. Этот подход позволяет создать копию репозитория на другом сервере или сервисе, чтобы иметь возможность восстановить данные в случае потери доступа к основному репозиторию на GitHub.

На GitLab зеркалирование [12] может быть выполнено через настройки репозитория. Чтобы создать зеркальную копию, нужно добавить удаленный репозиторий с помощью URL-адреса Git-хостинг-сервиса, с которого пользователь хочет зеркалировать свой репозиторий. После этого GitLab будет автоматически обновлять зеркальную копию репозитория при каждом обновлении исходного репозитория. GitLab также позволяет настроить зеркалирование для веток и тегов, а также позволяет добавить автоматические операции в зеркальную копию, такие как запуск CI/CD пайплайна при обновлении зеркальной копии. Данный подход полностью решает поставленную проблему, но GitLab зеркалирование является платной функцией.

Кроме GitLab функцию зеркалирования поддерживают и другие подобные сервисы, например, Codeberg [5]. Однако с 2020 года эта функция отключена, так как “Зеркальные репозитории легко создаются и постоянно растут, после создания они занимают ресурсы навсегда” [6].

Зеркалирование - это эффективный способ создания резервной копии репозитория. Однако, на данный момент все популярные сервисы либо не предоставляют доступ к функции зеркалирования, либо могут предложить её только за дополнительную плату. Также возможны ограничения на размер копии, доступное место для хране-

ния, а также на количество зеркалированных репозиториев на одной учетной записи.

1.3 Импортеры

В случае необходимости сохранения данных организации GitHub, можно воспользоваться специальными решениями, позволяющими импортировать содержимое репозиториев и метаданные.

Один из таких инструментов это python-github-backup [21]. Данный инструмент позволяет выгрузить содержимое репозитория по всем веткам с помощью зеркалирования, также он импортирует большую часть метаданных репозитория, но возникают проблемы с данными pull requests. Часть комментариев, а также reviews не импортируется. Кроме того та часть метаданных, которая связана с организацией, не может быть импортирована. Еще одна проблема заключается в размере копии, так как большая часть импортированных данных не нужна для восстановления, она становится избыточной и занимает место без нужды.

Другой подобный инструмент это GitHub-Backup [10]. Этот инструмент также позволяет импортировать содержимое репозиториев, однако только главную ветку, также он позволяет копировать часть метаданных репозитория, но есть ряд недостатков. Например, все pull requests импортируются и храняться как issues, а информация о reviews вообще не импортируется. Кроме того, как и в предыдущем инструменте, часть метаданных, связанная с организацией, не может быть импортирована.

В итоге, как правило, импортирование метаданных не происходит полностью, из-за чего возникают определенные проблемы при восстановлении. Кроме того, процесс восстановления не автоматизирован и может потребовать значительных усилий и времени со стороны пользователя.

1.4 Rewind

Другим подходом к решению проблемы сохранности данных на GitHub является использование инструмента Rewind [23].

Rewind backup для GitHub - это инструмент, который позволяет создавать резервные копии всех данных организации на GitHub, включая метаданные и содержимое репозиториев. Также инструмент Rewind предоставляет возможность быстро и легко восстанавливать все данные, включая все изменения, которые были сделаны в репозиториях на GitHub.

С помощью Rewind можно автоматически создавать резервные копии всех репозиториев и метаданных, сохранять эти данные в облаке и быстро восстанавливать их в случае потери или непредвиденного сбоя на GitHub.

Однако в процессе тестирования данного инструмента был выявлен ряд недостатков:

- Запуск возможен только для владельцев организации
- Восстановление возможно только пока сохранен доступ к исходной организации
- Если невозможно восстановить какие-то данные, инструмент не восстанавливает никакие данные
- Все pull request восстанавливаются как issues
- Вся специфическая для pull request информация, например reviews, не восстанавливается

Несмотря на эффективность данного инструмента, он является платным, что может затруднить его использование для некоторых организаций с ограниченным бюджетом.

1.5 Выводы

Был проведен анализ всех известных подходов к решению проблемы сохранности данных GitHub организации. Первый вариант это регулярные миграции на другой сервис. Такой способ может быть полезным на случай потери доступа к организации. Однако пользователь окажется заперт на другом сервисе, так как миграция обратно на гитхаб не так тривиальна. Другой подход это зеркалирование. Этот подход решает поставленную проблему, однако на данный момент на всех популярных сервисах данная опция либо платная, либо больше недоступна вовсе. Остальные подходы относятся к резервному копированию и восстановлению. Существуют решения, позволяющие импортировать данные организации, включая содержимое репозиториев и метаданные, однако часть метаданных импортируется некорректно, либо не импортируется вовсе, а также восстановление в таком случае не автоматизированно. Последний аналог это инструмент Rewind. Он позволяется как копировать, так и восстанавливать все данные, но он платный, что затрудняет его использование.

2 Предлагаемый подход

2.1 Требования

После изучения существующих подходов к решению, был сформирован следующий список дополнительных требований:

1. Копирование и восстановление – отдельные инструменты

   Разработка инструментов для копирования и восстановления должна проводиться независимо друг от друга, поскольку уже существуют инструменты для импорта данных с GitHub. Это позволит пользователям продолжать использовать свои инструменты для

копирования, а мой инструмент для восстановления, если им достаточно функциональности их текущих инструментов.

2. Регулярное резервное копирование

   В задачу разработки входит также настройка инфраструктуры вокруг разработанного инструмента. Важно обеспечить удобство и доступность настройки регулярности резервного копирования, чтобы пользователи могли легко и гибко настраивать, как часто нужно создавать копии данных, а также настраивать другие параметры, например, периодическую очистку старых резервных копий

3. Минимизация неактуальных уведомлений

   Необходимо учитывать, что определенные действия на GitHub могут вызывать отправку уведомлений, поэтому важно найти способы уменьшения количества неактуальных уведомлений, чтобы не создавать спам.

4. Возможность расширения на большее количество метаданных

   Так как на данный момент минимально требуется поддержать только основные метаданные, при разработке необходимо максимально упростить процедуру поддержки новых типов метаданных.

5. Возможность расширения на другие GitHub-подобные сервисы

   Данная работа актуальна не только для сервиса GitHub, но и других подобных сервисов, например, GitLab. Предполагается, что в дальнейшем одним из вариантов масштабируемости решения будет поддержка других сервисов. Поэтому необходимо упростить процедуру поддержки работы с другими GitHub-подобным сервисом.

2.2 Метаданные

Перед тем как перейти к разработке решения необходимо уточнить список метаданных, которые необходимо поддержать в первую очередь. Метаданные - информация, связанная с проектом или организацией. Примерная структура метаданных организации представлена на изображении (см. Рис. 1).

Очевидно, что разные метаданные имеют разную частоту использования. Я выделила основную часть метаданных, это информация об участниках, issues, pull requests и actions, эти метаданные содержит практически любая активная гитхаб организация.

При выделении основных типов метаданных я руководствовалась:

1. Опыт знакомых разработчиков
2. Данные, на которые чаще опираются в исследованиях
   - Can we make it better? Assessing and improving quality of GitHub repositories [2]
   - Characterization and Prediction of Popular Projects on GitHub [17]
   - A Large Scale Study of Long-Time Contributor Prediction for GitHub Projects [18]

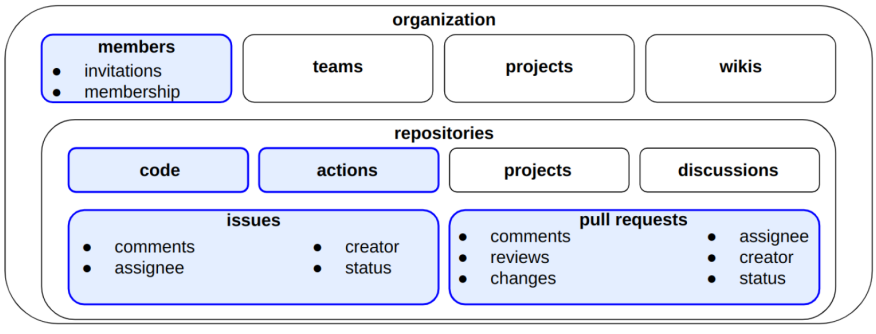

Рис. 1: Структура GitHub организации

2.3 Выбор оптимального подхода

Существует два основных вида резервного копирования [3]:

1. Полное резервное копирование

   При полном копировании все данные копируются сразу, без учета того, были ли они изменены или нет. Это означает, что полное копирование может занять много времени, а данные копии будут занимать много места, особенно для больших объемов исходных данных. Однако, когда требуется восстановление данных, процесс будет быстрее, так как все данные уже находятся в одном месте.

2. Инкрементальное резервное копирование

   Инкрементальное копирование копирует только изменения, произошедшие с момента последнего копирования. Это позволяет сократить время и объем, затраченные на резервное копирование, так как не нужно копировать все данные каждый раз. Однако, когда требуется восстановление данных, процесс может занять больше времени, так как для восстановления необходимо объединить несколько копий данных.

В целом, полное копирование более простое и быстрое для восстановления, но может занять много времени и места. Инкрементальное копирование, в свою очередь, более экономично в плане времени, потраченного на процесс копирования, и пространства необходимого для хранения резервной копии, но может потребовать больше времени для восстановления. Выбор между полным и инкрементальным копированием зависит от потребностей конкретного проекта и доступных ресурсов.

Был принят выбор в пользу полного резервного копирования, чтобы решить проблему сохранности данных GitHub организаций,

поскольку в данном случае скорость восстановления имеет больший приоритет, чем скорость копирования. В то же время, проблема большого объема данных может быть решена с помощью сжатия, что не является критическим фактором.

Итоговый алгоритм работы инструментов (см. Рис. 2):

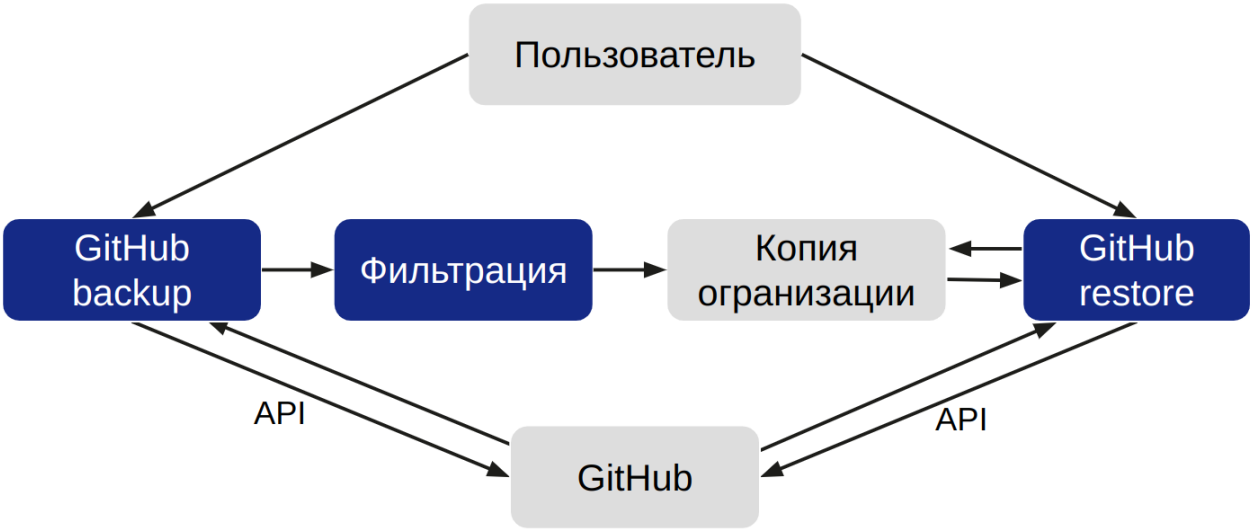

Рис. 2: Схема работы

При копировании, модуль копирования через GitHub API [9] извлекает все необходимые данные с сервера GitHub, выполняется их фильтрация при необходимости, после чего данные сохраняются.

При восстановлении, модуль восстановления проходит через эту коллекцию данных и, используя GitHub API, создает структуры, идентичные исходным.

2.4 Инструмент для копирования

Для выполнения резервного копирования организации необходимо выполнить два основных этапа. На первом этапе необходимо импортировать содержимое репозиториев. Простым решением является клонирование репозитория, однако в этом случае сохраняет-

ся только главная ветка, что недостаточно. Для того чтобы сохранить данные всех веток, было принято решение о создании зеркала репозитория, которое позволит импортировать содержимое репозитория и все активные ветки. Удаленные ветки не сохраняются, поскольку они потенциально содержат мало полезной информации, их может быть много, что затрудняет их хранение. Однако, зеркало имеет недостаток: при восстановлении, если основная ветка имеет название, отличное от ”master”, то будет назначена случайная ветка в качестве основной. Чтобы избежать этой проблемы, важно сохранить название основной ветки отдельно. Кроме того, следует сохранять дополнительную информацию о репозитории для более удобного восстановления.

Второй этап — резервное копирование метаданных с помощью GitHub API. Для работы с GitHub API есть библиотека PyGitHub [20], но она покрывает не все возможности API, которые были необходимы для реализации инструмента копирования. Также есть потребность более тонко настраивать работу с сервисом, например для обработки ошибок. Есть некоторые методы API, имеющие неочевидную логику возвращаемого кода. Например, в большинстве методов код 403 возникает при превышении rate limit, однако в некоторых методах этот же код отвечает за попытку получить доступ к данным, для которых недостаточно разрешений. Такие случаи хочется обрабатывать по-разному, библиотека PyGitHub этого не предусматривает. Поэтому было решено самостоятельно реализовывать методы взаимодействия с API. В результате были поддержаны следующие методы API:

- GET List organizations
- GET List organization members
- GET Get organization membership for a user

- GET List organization repositories
- GET Get a repository
- GET List repository issues
- GET List repository pulls
- GET List issue comments
- GET List reviews for a pull request
- GET List comments for a pull request review
- GET Get rate limit status for the authenticated user

2.5 Структура хранения

Для хранения импортированных данных была необходима разработка структуры хранения. Учитывая, что инструмент для копирования и восстановления предполагались как независимые друг от друга инструменты, возможность использования другого инструмента для импорта данных была учтена. В связи с этим было рассмотрено несколько вариантов структуры хранения данных.

Первый способ предполагает структуру, повторяющую структуру API GitHub с полным сохранением всех данных полученных в ответ на запросы (см. Рис. 3). Этот метод предоставляет удобство для пользователей, так как структура API GitHub широко используется и благодаря наличию документации является просто поддерживаемой. Кроме того, такой подход обеспечивает простую поддержку изменений API.

Однако, это приводит к хранению большого объема ненужной информации, так как для восстановления не требуется большая часть данных, которые возвращает GitHub.

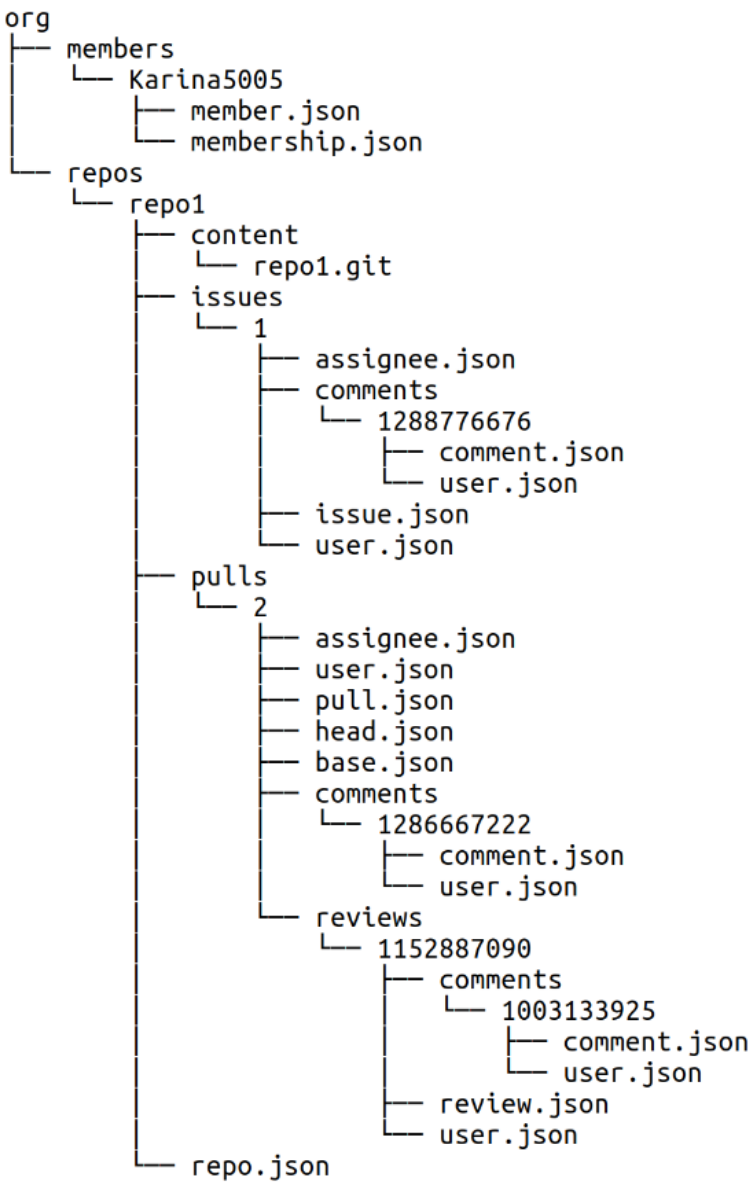

Рис. 3: Структура хранения, повторяющая структуру GitHub API

Второй подход предполагал создание собственной структуры хранения (см. Рис. 4). Он имеет преимущество в хранении минимального количества данных, необходимых для восстановления. Кроме того, это приводит к наиболее простому подходу к восстановлению, так как структуру можно легко настроить под восстановление. Но в таком случае возникает высокая связность между моделями копи-

рования и восстановлением, а также GitHub API, изменение любого компонента может привести к изменению всех остальных, что является критичным недостатком в процессе разработки и дальнейшей поддержки инструментов.

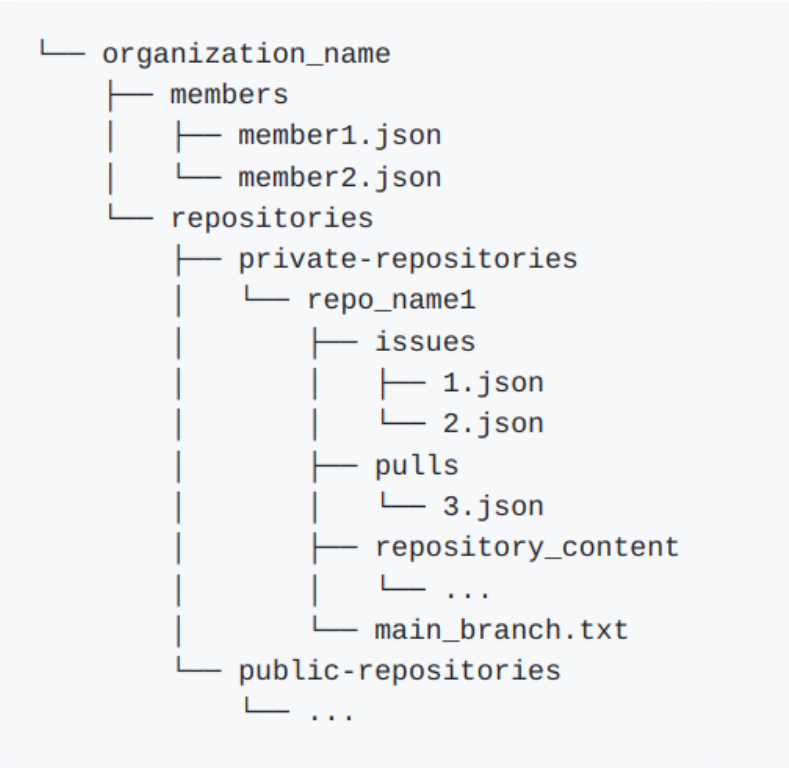

Рис. 4: Собственная структура хранения

Изначально был выбран второй подход, чтобы сократить объем импортированных данных. Однако, после анализа недостатков, было принято решение переписать всю работу с использованием первого подхода.

2.6 Фильтрация

Выбранная структура хранения для импортированных данных содержит значительную проблему с большим размером. Сохраняет-

ся большое количество ненужной информации, что приводит к большому размеру файлов. Ответы API GitHub содержат огромное количество информации, которая либо не нужна для восстановления, например, идентификаторы или ссылки, либо дублируется, например, для issues GitHub возвращается информация о создателе не только как логин или идентификатор пользователя, а как полный список данных об этом пользователе.

Чтобы решить эту проблему, было принято решение не менять структуру полученных ответов, а отфильтровать ненужные поля. На изображении (см. Рис. 5) показан пример фильтрации полей для пользователя, кроме логина и идентификатора, для восстановления никакие другие поля не нужны, поэтому их можно не сохранять.

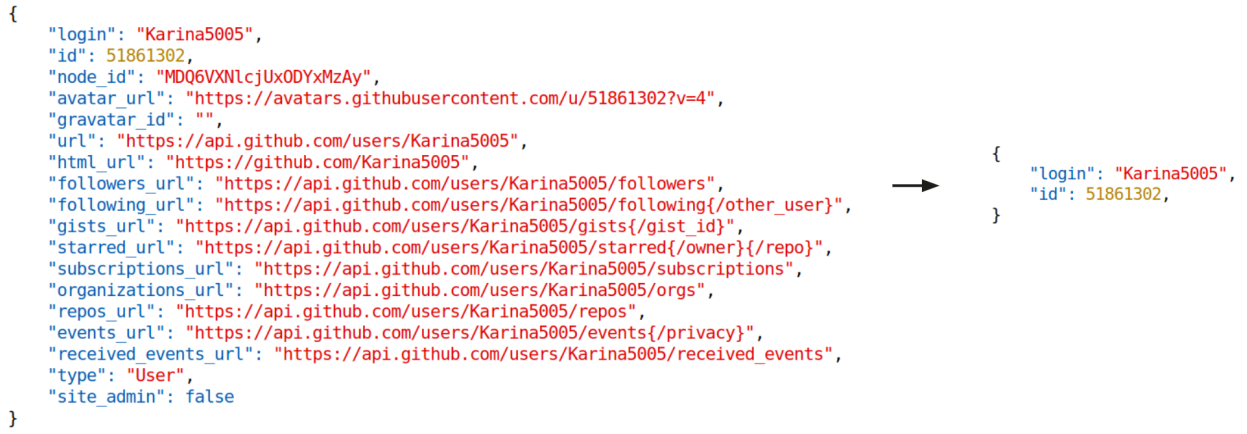

Рис. 5: Пример фильтрации

Таким образом, сохраняются все преимущества использования структуры GitHub API, поскольку модуль восстановления будет готов работать с полными файлами, не прошедшими этап фильтрации, но при этом будет извлекать только необходимые поля. Результатом этого подхода является небольшой размер копии, который всего на немного больше, чем при использовании собственной структуры хранения, но при этом сохранение всех преимуществ использования выбранной структуры хранения.

2.7 Инфраструктура инструмента резервного копирования

После завершения разработки модуля резервного копирования, я перешла к настройке инфраструктуры, что потребовало написания Ansible playbook [1]. С его помощью можно автоматизировать настройку сервера, установить все необходимые зависимости, настроить Cron job для запуска инструмента резервного копирования, архивировать полученные данные и логи, а также удалять устаревшие резервные копии спустя определенный промежуток времени.

Кроме того, после каждого запуска модуля копирования, набор метрик отправляется на Prometheus сервер, их можно использовать для отображения информации в графическом виде с помощью Grafana [15] (см. Рис. 6).

На данный момент в список метрик включены:

1. успешность резервного копирования
2. последний код выхода
3. дата последнего резервного копирования
4. размер занятого пространства

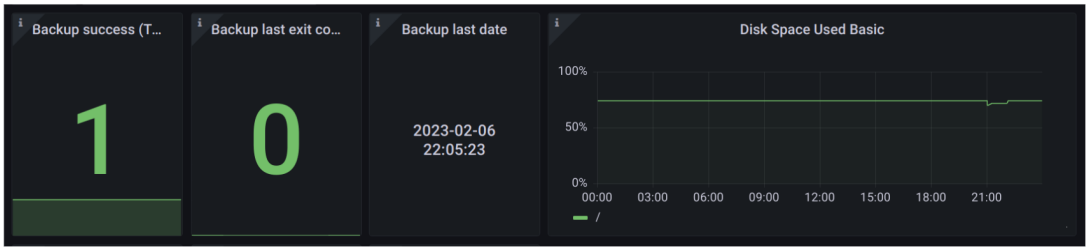

Рис. 6: Отображение метрик

Таким образом, полученные метрики помогают отслеживать состояние системы регулярного резервного копирования и обеспечи-

вать ее стабильную работу.

2.8 Инструмент для восстановления

Процесс восстановления также разделен на два этапа: восстановление содержимого репозиториев и восстановление метаданных.

Я приступила к работе по восстановлению организации, начав с восстановления содержимого репозиториев. Используя зеркало и всю необходимую метаинформацию о репозитории, в первую очередь создается репозиторий с нужными настройками с помощью GitHub API. Затем, с помощью Git, он заполняется содержимым. В итоге получается полностью идентичный репозиторий со всеми активными ветками и идентичной кодовой базой. Единственное отличие заключается в том, что удаленные ветки отсутствуют полностью, восстановить их нельзя.

Следующий этап это восстановление метаданных с помощью GitHub API. Библиотека PyGitHub [20] опять же покрывает не все необходимые для моей работы методы, поэтому было решено самостоятельно реализовывать методы взаимодействия с API. В результате были поддержаны следующие методы API:

- POST Create an organization invitation
- POST Create an organization repository
- POST Create an issue
- POST Create a pull
- POST Create an issue comment
- POST Create a review for a pull request
- POST Create a review comment for a pull request

- GET List branches
- GET Get a commit

Первым шагом этапа восстановления метаданных было восстановление участников организации. На этом этапе возникла проблема, GitHub не позволяет просто добавить участника в организацию, взамен предлагается система приглашений, поэтому наиболее разумным решением в данном случае является отправка приглашений участникам. Хотя это может показаться не совсем удобным, но другого варианта нет, даже при ручном восстановлении участников обойти этап рассылки приглашений невозможно. Как только участники примут приглашения, набор участников будет восстановлен, их роли и статусы будут такими же, как в исходной организации.

Следующий шаг заключается в восстановлении issues. Восстанавливаются все типы issues - открытые и закрытые, их статусы сохраняются. При восстановлении issue, восстанавливается все содержимое, включая тело задачи и все комментарии. Поскольку GitHub не предоставляет возможностей для восстановления, под восстановлением предполагается создание идентичного. Все данные создаются заново от имени пользователя, который предоставил свой токен для запуска процесса. Этот пользователь должен обладать всеми необходимыми правами. В комментарии к каждому issue указывается информация о его первоначальном создателе и том, кто был назначен на его выполнение. Было важно учесть рассылку уведомлений, которые генерируются при назначении issue на пользователей, поэтому было решено отправлять уведомления только об issues, которые еще активны, чтобы не создавать спам в виде уведомлений об уже закрытых задачах. Если issue открыто, то пользователь назначается на выполнение восстановленного issue, а если оно закрыто, то создается комментарий с информацией о том, кто решал эту задачу и кем она была закрыта в итоге.

Восстановление pull requests оказалось наиболее сложной задачей. Первым шагом было восстановление открытых pull requests. Этот шаг не вызвал особенных сложностей, через GitHub API удается восстановить всю структуру в исходном виде. Внутри pull request восстанавливается все содержимое, включая все комментарии, изменения и reviews.

Восстановление закрытых pull request в первозданном виде невозможно, так как внутри GitHub pull request это слияние двух веток, то есть коммитов, которые являются последними в ветках. Для закрытых pull requests это является проблемой, так как последние коммиты в ветках меняются, если эти ветки используются после слияния.

Первым возможным решением было восстановление pull request как issues. Такой стратегии придерживается инструмент Rewind. Однако такое решение неидеально, так как pull request содержит много дополнительной информации, отличной от issues:

- Commits
- Changes
- Reviews
  - reviews comments
  - reviews positions
  - reviews status

Все эти данные можно восстановить как комментарии, такой подход уже лучше, чем не восстанавливать эти данные вовсе, но большая часть из их этих данных в таком случае теряет наглядность и перестает иметь смысл. Поэтому появилась задача восстановления как можно большего количества pull request в максимально приближенном к исходному виде.

Для решения этой задачи был реализован алгоритм создания pull request из коммита в коммит (см. Рис. 7):

1. Нахождение коммитов, между которыми происходило слияние - head и base
2. Создание фиктивной ветки из коммита head - new head

   Последний коммит новой ветки содержит всю информацию об исходной ветки до коммита head
3. Создание фиктивной ветки из комита base - new base

   Последний коммит новой ветки содержит всю информацию об исходной ветки до коммита base
4. Создание pull request из ветки new head в new base

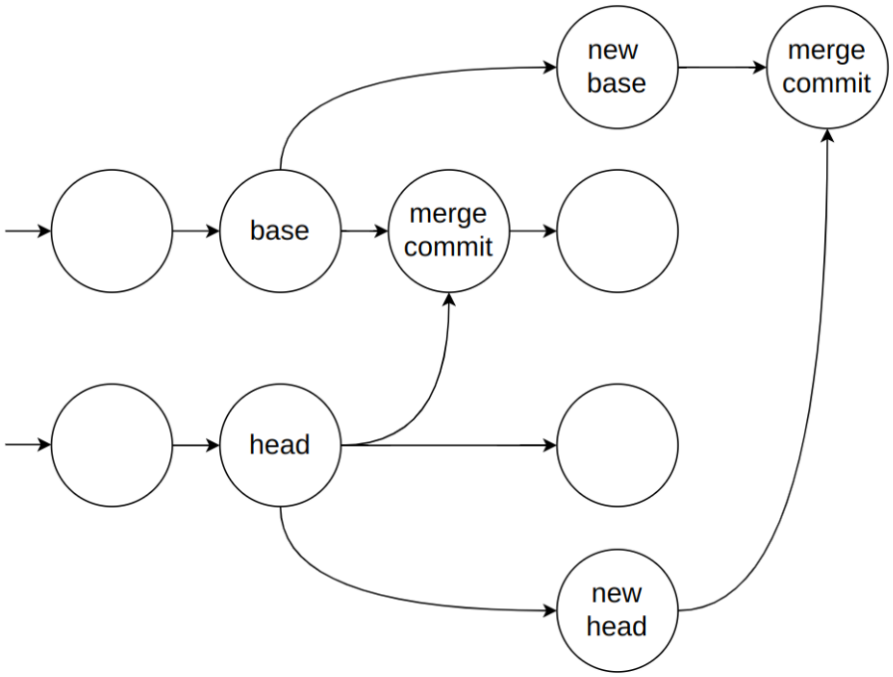

Рис. 7: Создание pull request из коммита в коммит

Применив такой алгоритм удается повторить полную структуру pull request, в особенности список коммитов и изменения.

Алгоритм восстановления pull request между коммитами уже является большим прогрессом в решении проблемы восстановления

pull request в первозданном виде. Однако, удаленные ветки могут оказаться проблемой при восстановлении pull request в исходном виде, так как необходимые данные о коммитах могут быть утеряны. Как было сказано ранее, удаленные ветки считаются избыточной информацией, поэтому они не восстанавливаются. В таком случае, ситуация удаления одной из веток pull request можно оказаться критичной, так как для восстановления не будет достаточно информации о коммитах. В этом случае возникает необходимость в дополнительных решениях для восстановления потерянных данных.

Хорошим сценарием оказывается ситуация удаление head ветки и слияние с создание merge commit. Такое слияние позволяет использовать merge commit в качестве head commit, поскольку он содержит те же изменения, что и исходный head commit. В итоге создается pull request из merge commit в base commit (см. Рис. 8). Такой подход позволяет восстановить практически идентичную структуру pull request, за исключением списка коммитов, которые невозможно восстановить, так как они принадлежат удаленной ветке. В этом случае будут восстановлены итоговые изменения, а также связанные с ними reviews и комментарии.

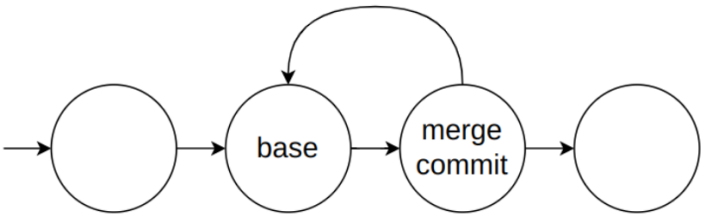

Рис. 8: Создание pull request из merge commit в base commit

Ситуации, когда происходит слияние без создания merge commit через rebase или же удаление base ветки, не являются оптимальными для восстановления pull request в первоначальном виде. В этих случаях восстановление pull request в исходном виде невозможно, поэтому используется альтернативный метод, описанный ранее, ко-

торый заключается в восстановлении данных в качестве issues. Это позволяет сохранить информацию, но не в том же формате, что и в исходном pull request.

2.9 Выводы

При разработке инструмента для восстановления данных с GitHub необходимо учитывать требования к разделению функций копирования и восстановления, регулярному резервному копированию, уменьшению количества неактуальных уведомлений, возможности расширения на большее количество метаданных и на другие GitHub-подобные сервисы.

Для реализации резервного копирования был выбран подход полного резервного копирования. Архитектура минимально завязана на GitHub, реализовав необходимые функции взаимодействия с GitHub API, можно поддержать и другие GitHub подобные сервисы. Для добавления нового типа метаданных достаточно написать методы для импорта и экспорта этих данных, опционально можно добавить функции для фильтрации данных.

Процесс резервного копирования организации делится на два этапа. Первый этап — резервное копирование репозиториев путем зеркалирования. Второй этап — резервное копирование метаданных с помощью GitHub API. Чтобы сделать структуру хранения простой и удобной для пользователя, было решено повторить структуру GitHub API. Также добавлена фильтрация, чтобы уменьшить размер резервной копии.

Процесс восстановления также делится на два этапа. Первый этап — восстановление гитовых данные благодаря зеркалу репозитория. Второй этап включает в себя восстановление метаданных за счет реконструкции идентичных структур GitHub с помощью GitHub API. На этом этапе критически важно было подумать об отказоустой-

чивости. Подход гарантирует восстановление как можно большего количества данных в первозданном виде.

3 Тестирование

3.1 Автотестирование

Стремясь повысить качество и эффективность полученных инструментов, в процесс разработки был внедрен процесс автоматизированного тестирования в модуле резервного копирования.

В процессе разработки был создан набор автоматических тестов, с помощью заглушек были написаны unit тесты для тестирования взаимодействия с GitHub API. Также написаны unit тесты и функциональные тесты, охватывающих основные функции и сценарии использования модуля резервного копирования. В результате модуль резервного копирования был покрыт автотестами на 90%.

Для модуля восстановления написание автотестов было проблематичным. С помощью заглушек были написаны unit тесты для тестирования взаимодействия с GitHub API. Тестирование сценариев использования модуля восстановления с помощью автотестов не представляется возможным, так как результат работы модуля требует субъективной оценки качества восстановления, которую может дать только человек при ручном тестировании.

3.2 Тестовая организация для базовых сценариев

Тестирование модуля восстановления автотестами практически невозможно, поэтому проводилось ручное тестирование. Для ручного тестирования была создана тестовая организация GitTstOrg [14], покрывающая все базовые сценарии использования организации. Для

ее создания проводилось моделирование процесса разработки внутри организации.

Содержимое организации:

- 2 участника
  - Owner
  - Member
- Открытое/Закрытое issue
  - Комментарии
  - Assignee
- Открытый/Закрытый pull request
  - Комментарии
  - Review
  - Комментарии к review
  - Assignee

Результаты тестирование инструментов на базовой организации представлены в таблице (см. Рис. 9).

	code	issues	pulls	reviews	members	size	time
копирование	+	+	+	+	+	191,2 kB	29s
восстановление	+	+	+	+	+	—	3m7s

Рис. 9: Тестирование на базовой организации

Тестирование показало, что реализованные инструменты справляются с задачей резервного копирование и восстановления всех основных данных GitHub организации в базовых случаях, восстановле-

ние полноценное, восстановленная организация имеет точное сходство с исходной.

3.3 Тестирование на реальной организации

Также проводилось тестирование инструментов на реальной организации Cloud Labs Shared Infrastructure [4], в которой 27 репозиториев в стадии активной разработки, что позволяет проверить большую часть возможных сценариев. Активными репозиториями считаются репозитории, имеющие хотя бы 1 созданный открытый pull request за последний месяц. Результаты тестирования представлены в таблице (см. Рис. 10).

	code	issues	pulls	reviews	members	size	time
копирование	+	+	+	+	+	6,7 MB	6m3s
восстановление	+	+	±	±	+	—	178m

Рис. 10: Тестирование на реальной организации

Есть проблема с восстановлением pull request и соответственно reviews, как было описано в главе 2.8, не каждый pull request удается восстановить в виде pull request, такие pull requests восстанавливаются в виде issues. Также есть проблема со скоростью работы модуля восстановления, обусловленная ограничением количества запросов к GitHub API [22].

3.4 Выводы

Реализованное решение протестировано, модуль копирования покрыт автотестами на 90%. Для ручного тестирования была создана тестовая организация и наполнена имитацией процесса разра-

ботки, в результате покрыты все базовые сценарии. Инструменты отлично показали себя при тестировании на тестовой организации. Также проводилось тестирование инструментов на реальной организации. Не все данные получается восстановить в первозданном виде, но все данные так или иначе восстанавливаются, что подтверждает работоспособность инструмента. Также есть проблема со скоростью работы, обусловленная ограничением количества запросов к GitHub API.

4 Сравнительный анализ

4.1 Инструмент копирования

Также был проведен сравнительный анализ существующих решений для копирования и полученного. Для этого использовалась небольшая тестовая организация GitTstOrg [14], покрывающая базовые сценарии. Тестирование возможно только на тестовой организации, так как для запуска части аналогов необходимо быть создателем организации. Результаты тестирования представлены в таблице (см. Рис. 11).

Можно заметить, что несмотря на небольшие потери в скорости работы, было достигнуто лучшее качество работы инструмента. В отличии от аналогов полученная реализация работает с организацией, а значит позволяет импортировать специфические метаданные, например, список участников. Также происходит более полное копирование данных о pull requests.

	code	issues	pulls	reviews	members	size	time
github-backup	+	+	+	+	+	191,2 kB	23s
python-github-backup	+	+	+	±	—	194,5 kB	8s
Rewind	+	+	±	±	—	45,1 kB	12s

Рис. 11: Сравнение инструментов копирования

4.2 Инструмент восстановления

Также был проведен сравнительный анализ существующих решений для восстановления и полученного. Среди аналогов для восстановления есть только Rewind. Тестирование возможно только на тестовой организации GitTstOrg [14], так как для запуска части аналогов необходимо быть создателем организации. Результаты сравнения представлены в таблице (см. Рис. 12).

	code	issues	pulls	reviews	members	time
github-backup	+	+	+	+	+	2m13s
Rewind	+	+	±	—	—	31s

Рис. 12: Сравнение инструментов восстановления

Реализованный инструмент предоставляет более широкие возможности для восстановления. Возможно восстановление большего количества специфических для организаций метаданных, например список участников. Также больше метаданных восстанавливаются в первозданном виде, например большая часть pull requests восстанавливается как pull requests, Rewind же все pull requests восстанавливает как issues. Кроме того, такой подход позволяет восстанав-

ливать reviews. Такой функционал аналоги не предоставляют вовсе. Но несмотря на эти преимущества, есть значительный недостаток в скорости работы полученного решения.

4.3 Скорость работы

Проблема со скоростью работы инструмента обусловлена ограничениями GitHub API. В базовых правилах использования GitHub API прописано требование на ограничение количество запросов в секунду, а именно не больше одного. При этом на POST запросы ограничение еще строже, кроме того они не фиксированы строго и GitHub оставляет за собой право эти ограничения изменять от запроса к запросу и с течением времени. Чтобы сократить количество блокировок, было решено сократить количество POST запросов в секунду до 0.2. Это увеличивает время работы инструмента восстановления, но в то же время сокращает количество блокировок на продолжительный период времени, что в итоге приводит к сокращению общего времени выполнения задач.

Таким образом, ограничения скорости работы инструмента вызваны требованиями, накладываемыми GitHub, чтобы гарантировать надлежащее использование ресурсов. Сокращение количества POST запросов и соблюдение этих ограничений помогает сократить количество блокировок, однако проблема со скоростью работы инструмента остается актуальной. В дальнейшем для решения данной проблемы с ограничениями скорости работы инструмента, связанными с GitHub API, можно применить следующие подходы:

- Оптимизация запросов

  Можно исследовать возможность объединения нескольких запросов в один или использовать функциональность пакетных запросов (batch requests), которая позволяет отправлять несколько за-

просов одновременно. Это может значительно снизить количество необходимых запросов, ускоряя работу инструмента.

- Кэширование

  Если данные не изменились с момента последнего запроса, можно использовать сохраненную версию данных из кэша, что поможет снизить количество запросов и ускорить работу инструмента, однако это не поможет решить проблему для POST запросов

- Управление ограничениями

  Для разных типов запросов, GitHub имеет разные ограничения. Тщательное изучение документации API для понимания ограничений и возможностей их управления может помочь повысить скорость работы. Также важно обновление инструмента в соответствии с изменениями ограничений.

4.4 Выводы

Тестирование инструментов проводилось на реальной организации, что позволяет проверить большую часть возможных сценариев. В результате тестирования было выявлено, что инструмент корректно отрабатывает на всех базовых сценариях использования GitHub организации.

Также был проведен сравнительный анализ существующих решений для копирования и полученного. Для этого использовалась небольшая тестовая организация, покрывающая базовые сценарии. Несмотря на худшее время работы, было достигнуто лучшее качество работы.

Также был проведен сравнительный анализ существующих решений для восстановления и полученного. В сравнении с аналогами реализованный инструмент справляется с полным восстановлением тестовой организации, но значительно проигрывает в скорости.

5 Заключение

5.1 Результаты

Для решения проблемы сохранности данных GitHub организации был выбран подход полного резервного копирования. В рамках этого подхода были разработаны и реализованы два инструмента: один для резервного копирования и импорта всей кодовой базы и основных метаданных, другой для автоматического восстановления этих данных.

Инструмент для резервного копирования позволяет пользователям безопасно сохранять всю кодовую базу и важные метаданные. Он обеспечивает полное импортирование основных метаданных данных: участники, issues, pull requests, гарантируя их сохранность и доступность в случае потери или повреждения данных организации.

Инструмент для восстановления, в свою очередь, позволяет автоматически восстановить всю кодовую базу и основные метаданные из созданных резервных копий. Это удобное решение, которое сокращает время и усилия, требуемые для восстановления данных, и минимизирует риск потери ценной информации.

Для проверки функциональности модулей была создана тестовая организация, которая была заполнена симуляцией процесса разработки. Это позволило убедиться в корректности работы инструмента восстановления и его способности вернуть кодовую базу и метаданные в состояние, идентичное сохраненным резервным копиям на базовых сценариях использования GitHub организации.

При сравнении полученного решения с аналогами было обнаружено, что оно обладает лучшим качеством работы. Восстановление данных происходит без потери информации, обеспечивая высокую степень точности и сохранность. Однако, при такой полноте

восстановления было замечено, что скорость процесса восстановления значительно меньше по сравнению с возможными аналогичными решениями. Эта проблема объясняется ограничениями GitHub API. Данная проблема была проанализирована, выделены возможные способы ее решения.

На данный момент инструмент, несмотря на некоторые недочеты, готов к использованию. Внутри компании Хуавей происходит внедрение реализованных инструментов в качестве постоянного решения для резервного копирования.

5.2 Дальнейшие планы

В перспективе в первую очередь планируется улучшить скорость работы, эта проблема не существенна для небольших организаций, но в масштабах больших организаций она может стать критичной.

Кроме того, есть планы расширить количество поддержанных метаданных, которые сохраняются во время резервного копирования. В настоящее время сохраняются основные метаданные, но будут добавлены дополнительные, например информация о дискуссиях, релизах и пр.

Кроме того, в планах расширение поддержки различных сервисов, подобных GitHub. Это позволит пользователям сохранять и восстанавливать кодовую базу и метаданные, используя не только GitHub, но и другие популярные платформы разработки, такие как GitLab, Bitbucket и т.д. Это расширение позволит разработчикам работать со своими проектами в любой выбранной ими системе, не ограничиваясь только одной платформой.

Эти планы расширения и улучшения позволят значительно повысить эффективность и функциональность инструментов резервного копирования и восстановления, обеспечивая более быструю и надежную работу с кодовой базой и метаданными, а также расширяя

возможности выбора сервисов разработки для пользователей.

Список литературы

1. Ansible. — 2023. — URL: https://www.ansible.com/.
2. *Azriadi G. A.* Can we make it better? Assessing and improving quality of GitHub repositories. — 2021.
3. *Chervenak A., Vellanki V., Kurmas Z.* Protecting File Systems: A Survey of Backup Techniques. — 1998. — Февр.
4. Cloud Labs Shared Infrastructure. — 2023. — URL: https://github.com/cloud-labs-infra.
5. Codeberg. — 2023. — URL: https://codeberg.org/.
6. Codeberg announcement. — 2023. — URL: https://blog.codeberg.org/mirror-repos-easily-created-consuming-resources-forever.html.
7. Gitea. — 2023. — URL: https://docs.gitea.io/en-us/.
8. Gitea-github-sync. — 2023. — URL: https://github.com/Muscaw/gitea-github-sync.
9. GitHub API. — 2023. — URL: https://docs.github.com/ru/rest?apiVersion=2022-11-28.
10. GitHub-Backup. — 2023. — URL: https://github.com/clockfort/GitHub-Backup.
11. GitLab. — 2023. — URL: https://docs.gitlab.com.
12. Gitlab mirroring. — 2023. — URL: https://docs.gitlab.com/ee/user/project/repository/mirror/.
13. Gitlab-github-migrate. — 2023. — URL: https://github.com/wollzelle/gitlab-github-migrate.
14. GitTstOrg. — 2023. — URL: https://github.com/GitTstOrg.
15. Grafana. — 2023. — URL: https://grafana.com/.

16. Import and migrate projects. — 2023. — URL: https://docs.gitlab.com/ee/user/project/import/.
17. *J. Han D. W., Yin J.* Characterization and Prediction of Popular Projects on GitHub. — 2019.
18. *L. Bao X. Xia D. L., Murphy G. C.* A Large Scale Study of Long-Time Contributor Prediction for GitHub Projects. — 2021.
19. Node-gitlab-2-github. — 2023. — URL: https://github.com/piceaTech/node-gitlab-2-github.
20. PyGithub. — 2023. — URL: https://github.com/PyGithub/PyGithub.
21. Python-github-backup. — 2023. — URL: https://github.com/josegonzalez/python-github-backup.
22. Rate limit. — 2023. — URL: https://docs.github.com/en/rest/rate-limit.
23. Rewind. — 2023. — URL: https://rewind.com/products/backups/github/.