

Moscow 2020

**Федеральное государственное автономное образовательное учреждение высшего
образования**

**«Национальный исследовательский университет
«Высшая школа экономики»**

Факультет компьютерных наук

Основная образовательная программа Прикладная математика и информатика

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ
РАБОТА**

на тему

Дэшборд для работы над Pull Requests

Выполнила студентка группы 166, 4 курса,

Токарева Елизавета Вадимовна

Руководитель ВКР:

Приглашенный преподаватель, Миловидов Алексей Николаевич

Оглавление

Оглавление	1
Аннотация	2
Abstract	3
Введение	4
1 Выбор и обработка данных для дашборда	6
1.1 Существующие решения	6
1.2 Данные для дашборда и настройки	7
1.3 Сбор данных	9
2 Хранение данных	12
3 Реализация веб-приложения	14
3.1 Страницы приложения	14
3.2 Инструменты	15
4 Развертывание на облачной платформе Heroku	16
5 Заключение	17
5.1 Дальнейшее развитие приложения	17
5.2 Итоги работы	17
Список литературы:	18

Аннотация

В настоящее время коды огромного количества проектов находятся на сайте <https://github.com>. Командам, работающим над такими проектами, необходим инструмент для удобного просмотра существующих пулл реквестов в свой репозиторий. В данной работе представлен дашборд, который позволяет значительно упростить работу для разработчиков и в ситуациях, когда им нужно быстро получить представление о текущем состоянии пулл реквестов, и в ситуациях, когда они хотят детально изучить открытые пулл реквесты, соответствующие некоторым их требованиям. В работе описывается реализация алгоритмов получения необходимых данных для дашборда при использовании GitHub API, описаны архитектура разработанного сервиса и процесс разработки его бэкенда. В результате работы созданная таблица была выложена на облачную платформу Heroku.

Abstract

Currently, codes for a huge number of projects are posted on the site <https://github.com>. Teams working on such projects need a tool for a systematic view of their repository's opened pull requests. This paper presents a dashboard that can significantly simplify the work for developers in situations where they need to quickly get an idea of the current state of pull requests, and in situations where they want to study in detail open pull requests that meet some of their requirements. The paper describes the implementation of used algorithms for collecting the necessary data for a dashboard using the GitHub API, the architecture of the developed service and the process of developing its backend. As a result of the work, the created dashboard was posted on the Heroku cloud platform.

Введение

В крупных проектах на гитхабе[1] код обновляется ежедневно, а для того, чтобы добавить свое изменение, разработчик отправляет пулл реквест в репозиторий проекта. Команда проекта и контрибьюторы проводят ревью открытых пулл реквестов, рекомендуют правки, ставят метки, просят других разработчиков тоже проверить код т.д. В самом пулл реквесте также много обновляемой информации: количество добавленных и удаленных строк меняется с добавлением нового коммита, статусы тестов. Чтобы не заходить по очереди во все открытые пулл реквесты и не проверять постоянно, были ли в них обновления, необходим инструмент для обзора открытых пулл реквестов и всей необходимой информации. Удобнее всего смотреть такую информацию в таблицах, поэтому целью диплома была разработка удобного дашборда.

Основными требованиями к дашборду являются:

- простота в использовании. Дашборд должен быть готов к использованию без дополнительных усилий со стороны пользователя.
- скорость работы. Обновление информации, собранной для дашборда должно происходить достаточно быстро, чтобы дашборд оставался удобным для использования и сильно ускорял работу. В большинстве случаев обновление данных должно происходить незаметно для пользователя. Загрузка страницы с уже обновленной информацией должна происходить со скоростью стандартной страницы из интернета.

- настраиваемость. Пользователь должен иметь возможность настроить дашборд так, чтобы пулл реквесты сортировались и фильтровались соответственно его требованиям.

Целевой аудиторией считаются команды разработчиков open source проектов, которым нужно следить за новыми пулл реквестами в репозитории и проводить ревью. За эталонный репозиторий был взят github репозиторий ClickHouse. Дашборд предназначен для использования этим проектом, но также может применен и к любому другому.

Работа состоит из четырех частей. В секции 1 рассматриваются существующие аналоги реализованного дашборда, анализируются данные, необходимые для получения наиболее полной информации о пулл реквесте из таблицы, и описываются алгоритмы их получения и обработки. В секции 2 обосновывается необходимость использования базы данных для хранения информации о репозитории и описывается ее использование, в секции 3 описывается работа с web для отображения таблицы, в секции 4 рассказывается о выкладывании дашборда на Heroku и об использовании шедулера событий, в секции 5 делаются выводы и описываются возможности дальнейшего развития дашборда.

Код проекта может быть найден по ссылке:
https://github.com/virbenka/pulls_dashboard, само приложение:
<https://pulls-dashboard-demo.herokuapp.com>

1 Выбор и обработка данных для дашборда

1.1 Существующие решения

Найти аналоги реализованного дашборда не удалось. В то же время, пулл реквесты всегда можно было смотреть на специальной странице гитхаба, а также существует дашборд[2], разработанный в Mozilla для целей, соответствующих теме диплома.

Filters	is:pr is:open	Labels 138	Milestones 0	New pull request				
129 Open	6,443 Closed	Author	Label	Projects	Milestones	Reviews	Assignee	Sort
[docs] minor translation update	pr-documentation	#11072	opened 1 hour ago	by blinkov				
Add's hasSubSeq function to match Array subsequences	doc-alert pr-feature	#11071	opened 3 hours ago	by r-zenine				1
Add storage rabbitmq read-only part	can be tested doc-alert pr-feature	#11069	opened 5 hours ago	by kssenii				
DOCS-112: any_join_distinct_right_table_keys	pr-documentation	#11065	opened 7 hours ago	by BayoNet • Draft				1
DOCS-323: LowCardinality	pr-doc-fix	#11060	opened 9 hours ago	by BayoNet • Draft				10
WIP Implement IDisk interface for HDFS	can be tested doc-alert pr-feature	#11058	opened 11 hours ago	by overshov • Draft				
Null data mods	doc-alert pr-feature	#11057	opened 13 hours ago	by Potya				
added simple regex		#11054	opened 15 hours ago	by Ill-phill-III • Draft				









































В таблице, предоставленной гитхабом, как плюс можно выделить очень качественно проработанную настраиваемость: сортировать и фильтровать можно по практически любым критериям. В то же время, не очень удобно, что стандартная сортировка выставлена как дата открытия пулл реквеста, хотя более интересна, как правило, дата обновления. Интерфейс можно назвать очень удобным для тех, кому хочется тщательно изучить конкретные пулл реквесты, но не очень подходит для тех, кому нужно получить

максимум информации просто окинув таблицу взглядом, потому что многие важные для пулл реквеста параметры (например, количество измененных строк) не отображаются или отображаются в неудобном для сравнения формате.

GitHub Pull Request Triage

Who is supposed to do what next on the open pull requests.

All open pull requests for github.com/yandex/ClickHouse

Bug	Title	Last updated	Assignees	Reviewers	People	Last actor	M*	S*	Changes	Labels
	11004: Remove allow_processors flag from executeQuery() -	23 minutes ago							 13 +353 / -92	
	11042: Apply TTL for old data, after ALTER MODIFY TTL query -	35 minutes ago							 12 +230 / -14	
	11032: DOCS-643: Sources of system metrics. +	36 minutes ago							 1 +34 / -5	
	11071: Add's hasSubSeq function to match Array subsequences -	37 minutes ago							 11 +281 / -36	 
	11036: Fixed parsing of S3 URL -	40 minutes ago							 2 +30 / -12	
	11065: DOCS-112: any_join_distinct_right_table_keys +	44 minutes ago							 2 +24 / -0	
	10973: Automatic generation of random datasets for a query -	an hour ago							 4 +1,213 / -0	  

Дашборд, созданный разработчиками Mozilla, в отличии от дашборда гитхаба, позволяет удобно сравнивать информацию, имеет названия колонок в шапке таблицы. В то же время, интерфейс все еще не интуитивно понятный, так как некоторые обозначения непонятны, а имена людей не подписаны. Отсутствует возможность фильтровать и сортировать пулл реквесты. К прочим минусам можно отнести долгую загрузку страницы.

Основываясь на данных, предоставляемых этими таблицами, а также на тщательном изучении составляющих пулл реквеста, можно получить некоторое представление о данных, которые необходимо предоставлять пользователю в дашборде.

1.2 Данные для дашборда и настройки

Столбцы в таблице состоят из:

- **Author**
Аватар, логин и отношение к репозиторию (Member, Contributor, ...) создателя пулл реквеста
- **Title**
Номер пулл реквеста, его название, описание
- **Status**
Названия тестов, их состояния (passed, pending, failed), время, в которое состояние обновлено
- **Labels**
Поставленный на пулл реквест метки
- **Mergeable**
Может ли пулл реквест быть влит в проект
- **Changes**
Количество коммитов, количество добавленных и удаленных строк
- **Assigned**
Люди, назначенные на коммит
- **Reviewed**
Люди, уже давшие ревью
- **Approved**
Люди, подтвердившие изменения
- **Last event**
Информация о последнем событии, произошедшем в пулл реквесте, аватар того, кто его совершил

Все ячейки содержат в себе ссылки на предоставляемую информацию.

Любую из колонок можно скрыть, указав такое предпочтение в настройках.

Параметры, по которым можно настраивать и фильтровать отображаемую информацию:

- **Сортировки**

- По дате обновления
- По дате создания
- По количеству пройденных тестов
- По размеру изменений
- По количеству комментариев

Все сортировки применимы и в порядке “по возрастанию”, и в порядке “по убыванию”

- **Фильтры**

- Показывать только пулл реквесты у которых прошли некоторые выбранные тесты
- Показывать только пулл реквесты, в которых участвуют определенные люди
- Показывать только пулл реквесты, авторы которых имеют выбранный статус
- Показывать только вливаемые (mergeable) пулл реквесты
- Показывать только пулл реквесты с не меньше/не больше чем выбранным количеством измененных строк

Фильтры можно комбинировать между собой и с сортировками

- **Количество**

Показывать только первые или последние n пулл реквестов. Сортировка в данном случае является сортировкой по времени обновления. Несмотря на это, к этому фильтру можно применять и другие (включая сортировки).

1.3 Сбор данных

Для сбора данных о пулл реквесте использовалось GitHub API. С его помощью можно собрать любую информацию, отправляя запрос на <https://api.github.com/<link>>. Полученный HTTP ответ можно преобразовать в json, из которого удобно извлекать все запрашиваемые данные.

Для авторизованного пользователя или приложения лимит запросов в час равен 5000. В связи с тем, что для сбора полной информации о каждом из пулл реквестов необходимо обратиться к API несколько раз, пришлось провести некоторую работу для уменьшения числа запросов. Одной из основных составляющих этой работы являлась постоянная проверка заголовка etag. В нем хранится идентификатор специфической версии ресурса. GitHub API не уменьшает оставшееся количество запросов в час, если значение заголовка запроса If-None-Match совпадает с etag заголовком ответа. Сравнение последнего полученного значения etag привело к заметным улучшением количества возможных обновлений данных репозитория, потому что, как правило, большая часть пулл реквестов репозитория или хотя бы их данных остается неизменными в разы больший промежуток времени, чем промежуток между обновлениями.

Бэкенд дашборда написан на языке программирования Python, поэтому для запросов использовалась популярная библиотека requests.

Несколько нетривиальным оказалось выявление последнего сделанного в пулл реквесте обновления. Время, получаемое по ключу last_updated оказалось не согласовано с данными ни из какого другого поля полученного json из этого или другого запроса. Поэтому варианты поиска последнего события состояли из:

- времени последнего сделанного коммита

- времени последнего оставленного комментария
- времени последнего ревью
- времени последнего события по данным из запроса на страницу

events: https://api.github.com/<owner>/<repo>/issues/<pull_number>/events

Оказалось, что время последнего событие со страницы events может быть больше, чем время обновления репозитория -- в таком случае событие не представляет интереса для дашборда (например, что кто-то упомянул этот реквест). Поэтому, чтобы отбросить такие события, при выборе последнего обновления не просто ищется самое позднее время среди возможных вариантов, а ищется наиболее близкое к значению поля `last_updated`.

Так как на каждый репозиторий может приходиться довольно много пулл реквестов (на момент написания этой работы количество открытых пулл реквестов в репозитории ClickHouse стандартно держится около 120), последовательная обработка могла бы занять значительное время. Чтобы избежать этого, после получения номеров всех открытых пулл реквестов сбор данных про них происходит посредством запуска по потоку на реквест. Потоки и ассоциированные с ним номера пулл реквестов складываются в очередь и сбор не заканчивается, пока очередь не опустеет.

Время ожидания окончания работы потока составляет 5 секунд, время ожидания HTTP ответов — 2 секунды (хотя для получения данных о репозитории время увеличено в два раза), количество повторных попыток получить ответы — 8. Параметры соответствуют выставленным в итоговом приложении, что не мешает менять их при локальном запуске кода.

2 Хранение данных

На определенном этапе разработке стало ясно, что данные нужно хранить, а не собирать информацию каждый раз заново. Причины:

1. **Скорость при формировании страницы без обновления базы.**

При хранении данных появляется возможность просто извлечь нужную информацию из базы, а не собирать их каждый раз заново для всех 120 рекестов. В абсолютном большинстве случаев извлечение данных из базы -- более удобное и быстрое действие.

2. **Скорость при формировании страницы с обновлением базы.**

Как правило, при большом количестве пулл рекестов все они не обновляются настолько часто, насколько пользователь хочет смотреть обновления. То есть даже собирая часть данных заново, можно делать меньше запросов -- например, если количество комментариев (его можно увидеть на странице с основной информацией по пулл рекесту) не изменилось, то отдельно

отправлять запрос на страницу, отвечающую за комментарии, не нужно.

3. **Уменьшение количество запросов в GitHub API.** Как уже было рассказано в предыдущей секции, снизить количество запросов можно при помощи проверка заранее сохраненного значения заголовка etag.

К минусам использования базы данных можно отнести то, что при первом открытии дашборда для конкретного репозитория (самым первым пользователем, к следующим за этим открытиям этого же репозитория даже с другого устройства проблема не относится) страница может загружаться довольно долго, так как помимо сбора данных в процесс обработки входит добавление всей собранной информации в базу данных, а потом извлечение из нее в подходящем формате.

Для хранения данных используется документная база MongoDB[6]. Выбор пал на нее из-за скорости и потому что для отображения веб-приложения удобно хранить данные документами, так, что их удобно извлечь в формате JSON.

3 Реализация веб-приложения

3.1 Страницы приложения

Приложение состоит из нескольких страниц:

Адрес	Исполнение	Отображение
/choice, /home	Вызывается функция, создающая форму, которую пользователь информацией о репозитории, который он хочет использовать. После исполнения перенаправляется на страницу с самим дашбордом.	Отображается как форма
/dashboard/<owner> /<name>	Вызывается функция, извлекающая из базы данных информацию по нужному репозиторию (с заданными параметрами), а если информации нет, то предварительно вызывается функция, собирающая и добавляющая информацию в базу.	Отображается как таблица
/dashboard/<owner> /<name>/settings	Вызывается функция, собирающая общие данные по репозиторию (например, какие тесты есть) и отправляющая пользователю форму для настройки по предложенным параметрам. После исполнения перенаправляется на страницу с самим дашбордом.	Отображается как форма

/dashboard/<owner> /<name>/refresh	Вызывается функция обновления данных по конкретному репозитория. После исполнения перенаправляется на страницу с самим дашбордом.	Не отображается, сразу перенаправление на таблицу
/task	Вызывается функция обновления данных по всем существующим в базе репозиториям. Если последний раз пользователь запрашивал дашборд с репозиторием более 5 дней назад, данные удаляются.	Не отображается, сразу перенаправление на таблицу

3.2 Инструменты

Приложение `pull_dashboard` было написано на языке Python с использованием фреймворка Flask[4] для работы с веб страницами. Данные в html страницу отправляются при помощи шаблонизатора Jinja2, а интерфейс реализован при помощи библиотеки flask-bootstrap, использующей Bootstrap 3 [3].

Время в шаблонах обрабатывается функциями из библиотеки Moment.js.

4 Развертывание на облачной платформе Heroku

В то время как приложение может быть запущено локально, многим пользователям удобнее использовать уже запущенный дашборд.

Одним из популярных решений для flask приложений является использование выбранного для дашборда Heroku[5] платформы. Для развертывания приложения необходимо было создать отдельную ветку в гитхаб репозитории и связать ее с Heroku аккаунтом.

Для работы с приложением в окружении должны быть заданы некоторые конфигурации (такие как гитхаб токен, название flask приложения и т.д.). Heroku дает возможность задать эти конфигурации через командную строку или же в настройках приложения на сайте

Также Heroku есть возможность создавать дополнительные (так называемые add-ons) элементы. Для приложения было использовано два add-on:

1. mLab MongoDB: позволяет создать MongoDB базу и подключить к ней приложение.
2. Heroku Scheduler: позволяет запускать выбранную команду раз в определенный промежуток времени (каждые 10 минут, каждый час или каждый день). Для приложения был выбран интервал в 10 минут. Каждый запуск Heroku шедулера открывает страницу `../task` для сбора обновлений в сохраненных репозиториях. Таким

образом пользователь, открывающий (уже добавленный в базу ранее) репозиторий, всегда получает информацию не старше чем 10 минут. При желании он может обновить данные нажатием на кнопку refresh. При локальном запуске аналогичный функционал дает изменение cron файла.

5 Заключение

5.1 Дальнейшее развитие приложения

Следующим шагом развития проекта могло бы стать улучшение плана на Heroku сервере для большей стабильности и частоты зашедуленных задач, исправление задержки при первом заходе в репозиторий. Также можно было бы провести работу по исследованию частоты запроса данных для репозитория и обновлять данные не каждый интервал времени, а более согласованно с потребностями (привело бы к уменьшению запросов к GitHub API). Кроме того, можно было бы создавать личные кабинеты пользователей с сохраненными “любимыми репозиториями” и настройками для них.

5.2 Итоги работы

Задачи дипломной работы была выполнена, было создано более удобное, чем имеющиеся аналоги, приложение с дашбордом для работы на пулл реквестами в репозиторий.

Список литературы:

1. GitHub Api docs

URL: <https://developer.github.com/v3/>

2. Mozilla dashboard code:

URL: <https://github.com/peterbe/github-pr-triage>

3. Bootstrap

URL: <https://getbootstrap.com/>

4. Flask docs

URL: <https://flask.palletsprojects.com/en/1.1.x/>

5. Heroku docs:

URL: <https://devcenter.heroku.com/>

6. MongoDB docs

URL: <https://docs.mongodb.com/manual>