This is a very simple example of a request you can make in the GitPrime API. Advanced features, such as Traversing Objects and Aggregating Data that allow you to customize your results, are also available.

Supported HTTP Methods

The GitPrime API provides read-only access to the objects, which means only GET requests are supported at this time.

Response Object

The response object contains the following information:

- **Count –** The total number of records available in the record set
- **Next** – The net URL in a pagination sequence; null If at the end
- **Previous** – The previous URL in a pagination sequence; null if at the beginning
- **Results** – The data of the request

Count, Next, and Previous are used for paginating large result sets describe below. However, all your action is in the results field.

Let's break down the URL. It's pretty straightforward.

Request URL

The request URL is a string representing the URL of the request. The URL syntax looks like this:

Request URL

```
https://app.gitprime.com/v3/customer/core/authors/?limit=3
```

[protocol]:// [hostname] [api version] [customer/core] [object] [parameter(s)]

Where:

- **Protocol** is the network protocol used to make the request: For security reasons, requests to the GitPrime API must always use the **https** protocol.
- **Hostname** is the IP address or URL for the GitPrime application. If you're using an on-premise version of GitPrime, this will be something other than **app.gitprime.com**.
- **API Version** refers to the version of the API being accessed.
- **/customer/core** will always be the same.
- **Object** is the resource in the GitPrime API from which you want to extract data. Examples are AUTHORS, COMMITS, and TARGETS. See Customer API for the complete list.

- **Parameter(s)** are the filters you use to extract a specific result, such as an Id, or a query parameter to restrict the result to a specific set of data. Parameter passing follows the widely used **?key=value&** pattern.

Supported HTTP Response Codes

The GitPrime API supports standard HTTP response codes.

Code	Description
200	**Success**. The request was successful. The results are displayed in the response body.
400	**Bad Request.** The request was unsuccessful. The problem is usually tied to incorrect request input. Error details are displayed in the response Body.
404	**Not Found.** The results are empty. No data matched your request.
500	**Internal Server Error.** The GitPrime system experienced an unexpected error. The problem has an unknown cause.

Supported Date and Datetime Formats

The GitPrime API supports ISO format date and datetimes.

For example:

2018-06-25 or **2018-06-25T00:00:00**

API Permissions

GitPrime provides object-level permissions.

Note that only Owners on a GitPrime account have access to manage API keys. You must grant any non-owner permissions to manage API keys. An example would be to give permission to a team lead to be able to distribute additional API keys for various integration projects.

In general, we recommend you create an API service account rather than mapping the API to an individual. Typically, you name that service account something generic like API_SERVICE_ACCOUNT.

View Rights and the API

View Rights are a key feature in GitPrime to control the depth of information a user can see in our interface. To learn more about View Rights in general, visit the GitPrime Help Center.

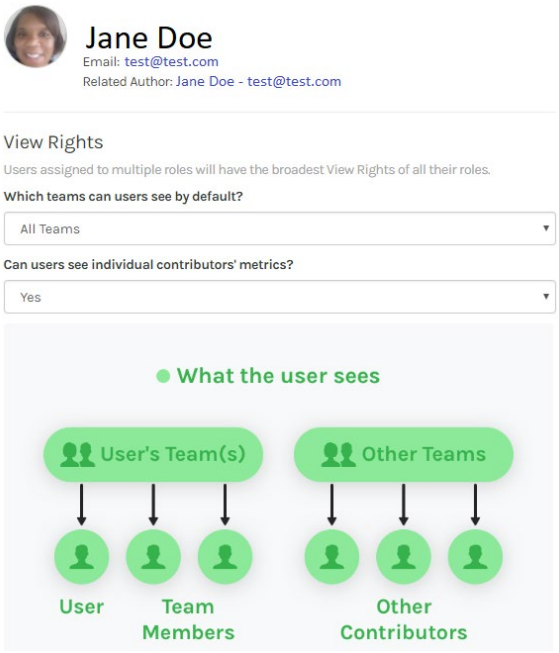
However, View rights are completely bypassed in the API. This is by design. View Rights are report dependent. The API is based on primitive objects, not reports. This is important to understand as it has serious security and information access implications. If you give a person access to the

COMMITS object in the API, for example, they will have complete unrestricted access to that object. That means they will be able to see any commit in any team in any repo.

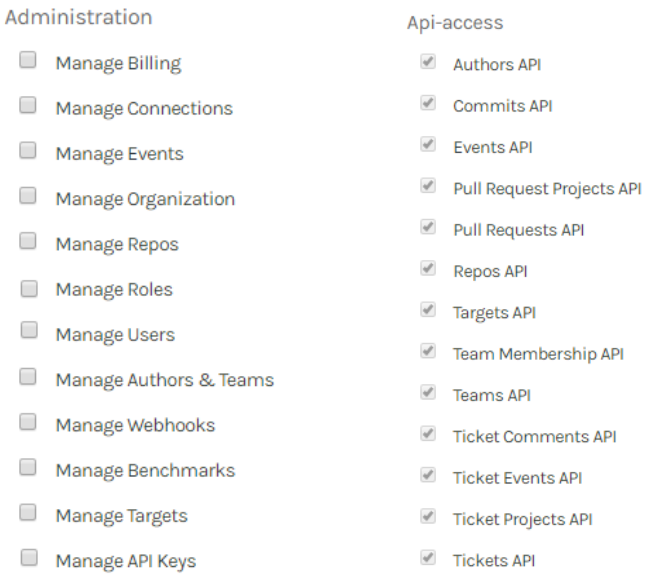
If you wish to restrict or control that access, you must enforce it at the client level. This is in part why we strongly recommend you use a service account.

Assign API Permissions

1. Click the avatar, located in the bottom-left of the screen, and then select **User Management** from the list that is displayed.
2. Select a user link. The user panel opens. An example of an existing user panel is shown in the following image:



3. Once you're there, select the API-related role(s) and permissions you want to assign to the user:
    - **Administration > Manage API Keys** – Create, assign, and deactivate API keys
    - **Api-access** – List of APIs (objects**)** from which users can request data. If the user will access the GitPrime API from a REST client or use an application to integrate GitPrime data, they will need an API key. See Authentication.



4. Click
5. **Save Changes**. That's all there is to it!

Interested in learning about all the GitPrime roles and permissions? Visit the GitPrime Help Center.

Aggregating Data

A common use case is generating results based on aggregating values as you would in SQL. Aggregation is a powerful function that returns a single value based on a set of values. You should be familiar with the GROUP BY clause in SQL to use this feature. GitPrime provides a robust REST-based syntax that emulates SQL aggregates.

For example, you may want the average churn over the last year or the largest commit ever. Normally, you would have to download all the COMMITS and perform the aggregate function locally in your client. Using GitPrime's built-in aggregate function you can accomplish this with a single call.

The GitPrime API provides aggregate functions to spare you from ingesting large quantities of data to perform your own aggregation. This functionality includes:

- Multiple aggregate functions in a single request
- Customizable grouping
- Ordering that includes aggregate fields

Aggregate functions that are currently supported:

- “avg” – Mean Average
- “count” – Count
- “max” – Maximum
- “min” – Minimum
- “std” – Standard Deviation
- “sum” – Sum
- “var” – Variance

Basic API Usage

Let’s look at an example of what a basic (but common) request might look like: we want to count the number of new lines of code in the last 30 days by Author “Bob Smith.”

In order to accomplish this without the benefit of aggregates, we would likely need to write a function that first collects all of Bob’s commit documents by iterating through numerous sets of results using limit and offset in individual GET requests. In a second loop, we’d then have to iterate through each commit document and sum the value of the “new_work” field. That is a lot of work and overhead in order to simply add up a single set of values.

Or you can use aggregates!

Request:

https://www.gitprime.com/v3/customer/core/commits?author__name=Bob%20Smith&date_created_gte={iso_date}

Response:

```
{
   “count”: 100,
   “next”: https://www.gitprime.com/v3/customer/core/commits/?limit=100&offset=100”,
   “previous”: null,
   “results”: [<full list of complete Commit documents>]
}
```

An aggregate request looks like this:

https://www.gitprime.com/v3/customer/core/commits?author__name=Bob%20Smith&date_created_gte={iso_date}&aggregate(sum)=new_work

With this response:

```
{
   “count”: 100,
   “results”: {
        “new_work_sum’: 1590
    }

}
```

You’ll notice that a few things are different:

The “.agg” suffix on our COMMITS URL resource tells the API that we are going to process this request as an aggregated request.

- The “aggregate” query parameter key defines which aggregate function to apply, while the value after the equal sign is the field to perform the aggregate function on.
- Our response document has changed. Our main “count” continues to reflect the number of rows that match our filter criteria, while “results” now contains only our aggregate calculations. In our example, you can see that Bob Smith had 100 matching COMMITS based on our 30-day filter criteria with a total “new_work” sum of 1,590 lines of code.
- The naming convention for fields in our aggregate “results” is {field_name}_{aggregate-function}. In our example, we performed a sum against the field “new_work”, resulting in the field name “new_work_sum.”

More Complex Aggregate API Example

Let’s say we wanted to see the count of COMMITS and average “new_work” lines of code per Commit for each Author on the team in the same 30-day window. We also want this data grouped by unique ID of the Author and the unique ID of the Repo.

Request:

https://www.gitprime.com/v3/customer/core/commits.agg?date_created_gte={iso_date}&aggregate[count]=id&aggregate[avg]=new_work&group_by[author_id,repo_id]

Response:

```
{
  “count”: 175,
  “results”: {
      {
        “author_id”: 1,
        “repo_id: 1,
        “id_count”: 25,
        “new_work_avg”: 560
      }
      {
        “author_id”: 1,
        “repo_id”: 2,
        “id_count”: 75,
        “new_work_avg”: 39
      }
      {
        “author_id”: 2,
        “repo_id”: 1,
        “id_count”: 50,
        “new_work_avg”: 14
      }
}
```

You can see from this example, there have been 175 total COMMITS by the two AUTHORS on our team. Author 1 has been working on both Repo 1 and Repo 2, while Author 2 is working only on Repo 1. Also, note that you can see that the average quantity and distribution of their “new_work” differs significantly.

Advanced Aggregates

These are relatively simple examples. You can use as many aggregate functions as you please and group by as many fields as you like based on the fields made available by each resource.

Additionally, ordering your aggregate results works in the exact same way as the conventional API calls, with the “ordering” query parameter and a comma-separated list of fields (fields prefixed with “-“ will order descending). If you wish to order by one of your aggregate fields such as “new_work_avg”, you can do that by saying “&ordering=new_work_avg”.

With this in mind, we could easily build more complex queries, for instance:

https://www.gitprime.com/v3/customer/core/commits.agg?date_created_gte={iso_date}&aggregate[count]=id&aggregate[sum]=new_work&aggregate[sum]=churn&aggregate[avg]=new_work&aggregate[avg]=churn&aggregate[max]=created_at&group_by[author_id]&ordering=id_count,-churn_sum

Which translates to:

“Give me the total number of commits, total lines of new work, total lines of churn, average lines of new_work per commit, average churn per commit and most recent commit date of each author, ordered by most total commits and highest average churn in descending order.”

Attempting to do this manually would’ve required a significant amount of development effort at a substantial performance cost. The GitPrime Aggregate API is designed to help provide insightful metrics with minimal pain.