
Bioinformatics Recipes

Aug 01, 2023

Table of Contents

1	FAQ - Frequently Asked Questions	3
1.1	Answers to reviewers	3
2	Installation	11
2.1	Running a Demo	11
2.2	Initialize Recipes	12
2.3	Start Server	12
3	Customize Settings	13
3.1	Directory Structure	13
4	Deploying Site	15
5	Projects	17
5.1	Project Privacy	18
5.2	Create a Project	18
5.3	Project Access	19
5.4	Granting Access	20
6	Data	23
6.1	Data Types	23
6.2	Upload Data	23
6.3	Import Directory	25
7	Recipes	27
7.1	Recipe Ingredients	27
7.2	Create a Recipe	27
7.3	Interface Specification	29
7.4	Interface Builder	29
7.5	Data Field	30
7.6	Recipe Template	31
7.7	Recipe Execution	32
7.8	Job Runner	32
8	Results	33
8.1	Output Directory	33
8.2	Rerun Results	33

9	Commands	35
9.1	Create a Project	35
9.2	Granting Access	36
9.3	Upload Data	36
9.4	Create a Recipe	37
10	API	39
10.1	Commands	39
10.2	Methods	40

Bioinformatics Recipes is a [Python](#) and [Django](#) based data analysis platform.

It is a simple, generic, flexible and extensible software that connects computational experts with end users.

FAQ - Frequently Asked Questions

1.1 Answers to reviewers


We created this section to answer questions reviewers posed while reviewing our scientific publication.

1.1.1 Who can run a recipe?

To run a recipe, the recipe must be **authorized**, and the user must have **trusted** designation.

1.1.2 What is an authorized recipe?

Since a recipe may contain shell commands and other code, security checks are needed to avoid the misuse of computational resources. Every new recipe starts in a so-called **pending authorization** state, displayed with an orange ribbon. A recipe with **pending authorization** cannot be executed on the website, but it may be inspected, viewed, shared, or downloaded.

Run :  Evaluate SNP Calling Accuracy

Pending authorization


This recipe evaluates SNP callers on a the syndip data.

Chromosomal Range:


The range of the data to be extracted

✓ Run

↺ Cancel

 This recipe needs to be audited by an admin before it can be run.

A user with administrative privileges (an administrator) must approve a recipe in the recipe edit window for the recipe to become executable within the website. A green ribbon decorates authorized recipes.

Run :  Evaluate SNP Calling Accuracy

✓ Authorized

This recipe evaluates SNP callers on a the syndip data.

Chromosomal Range:

The range of the data to be extracted

✓ Run

↺ Cancel

1.1.3 Who is a trusted user?

Each user has a designation: **trusted** or **visitor** that controls their ability to run recipes. only users with **trusted** designation may run **approved** recipes.

1.1.4 How do I become a trusted user?

The site administrators can change the designation of any user.

The restrictions that we have in place provide high granularity control of the computational resources.

The owner of the site decides which users and which recipes gain the privileges to use the computational resources. Other groups running the Recipes software may set up their system in a manner to automatically trust every new user that signs up, and they may also choose to approve every recipe that is created automatically.

1.1.5 How can I tell if I am a trusted user?

Users can find out what their designation is by looking at their user name in the menubar.

[? About](#)[Bioinformatics Admin](#)[Logout](#)

Each type of user gets different icons. These icons are used to indicate their designation.



Visitor



Trusted



Admin

1.1.6 Will this software run on my machine?

The software was designed with decentralization in mind. The software runs on any operating system: Linux, macOS and Windows 10 (with Linux Subsystem), and on any hardware that supports Python. We routinely run the software on a MacBook Air laptop, on a single computer serving a lab, and on a high-performance multicore server.

Installation takes little more than minutes and requires no special software, just support for the Python programming language. We do envision different groups running their personalized instances of the software to serve local needs.

1.1.7 Is the software useful in bioinformatics education?

Even though initially the platform was designed to provide bioinformatics support to our biologist collaborators, we have found that the use of recipes integrates exceedingly well within bioinformatics curricula.

Specifically, when covering more advanced topics, educators typically present a series of commands that demonstrate a chain of data analysis steps. Currently, there is no straightforward way to publish both the code and the results, all in one location. We can use say GitHub to publish code, but we can't use GitHub to execute the code, and we can't use GitHub to store large datasets either.

In contrast, when using Bioinformatics Recipes, students can see both the code and all the files and results generated while running the code. In addition, they can also view results generated with different runtime parameters; all results are linked to the recipe that produced them.

Finally, students can readily copy the recipe over to their projects, make changes to it and see the results of their changes, all in the same interface. Then they can download the recipes onto their systems and run the recipe within their environments.

1.1.8 The software seems conceptually most similar to Galaxy, but how does it differ?

When compared to Galaxy, the Recipes framework presents several fundamental differences in its operating principles:

1. The code downloaded from the site can be run on any other computer (that has the software installed) and will produce the same output as seen via the web interface.
2. A Galaxy analysis only runs inside the Galaxy software. Recipes are designed to be shared and expanded upon by various users.
3. Users may create different interfaces for recipes copied from someone else. The end-user cannot change interfaces in Galaxy.

4. The framework is also a data analysis know-how, social interaction, and training material distribution framework.

1.1.9 What are the advantages of the recipes over Galaxy?

One significant advantage, in our opinion, is the independence of the method from the platform.

A bioinformatics recipe is an independent piece of code that educates and trains bioinformatician how to perform the analysis themselves on any computational platform. The code may be run either on the Recipes website or on any other computational infrastructure. The code obtained from recipes is identical to the code a bioinformatician would develop at the command line.

Additionally, the platform also serves as a knowledge distribution. Users can build upon each others' know-how and expertise. A user may take an existing multistage analysis and add/remove/customize that analysis for their needs.

1.1.10 If you have a local Galaxy server and a local Recipe instance - what functional differences do users see?

It is possible to set up recipes that look and behave like tools in Galaxy. For example, if one were to wrap individual tools into recipes, then from the usability perspective, Galaxy and Recipes would be very similar.

In that sense, the Recipes website is a superset of some of Galaxy's functionality. Let us point out that the described approach is not how we envision using the Recipes. We advocate building pipelines rather than running separate tools, a strategy that is not directly applicable in Galaxy.

Finally, users may also choose to build different interfaces for the same data analysis pipeline. Users may choose to customize additional parameters. For example, suppose a user publishes a recipe that runs a short read aligner with default settings on a hardcoded fraction (say 25%) of a data:

Run: Short Read Alignment

✓ Authorized

The recipe demonstrates the use of different short read aligners.

Accession number for the reference genome:

AF086833

Must be an NCBI accession number


SRA run number:


SRR1972739

Must be an SRR run id number

✓ Run Cancel

Another user may take the same recipe, keep the same code for it, but create a different interface. They may choose to expose the subsampling percentage as a parameter:

Run :  Short Read Alignment (with subsampling)

 Authorized

The recipe demonstrates the use of different short read aligners on a random sub-sampling of the input reads.

Accession number for the reference genome:



Must be an NCBI accession number

SRA run number:

Must be an SRR run id number

Subsampling (percent):

Enter a percentage for random sampling.

 Run
 Cancel

By building on the work of another scientist, the user was able to create another pipeline, that now offers the extra functionality that they needed.

1.1.11 How does the performance compare?

Our framework uses Python 3 and was built with the [Django](#) application server, a well-documented platform with extensive use in the information technology industry. Django runs platforms such as Pinterest, Instagram, and many others. [Django](#) as an application development platform is well documented. Knowing Django is also a valuable skill that adds to the marketability of bioinformaticians.

When it comes to the performance of the analyses themselves, these depend on the choice of methods and on the infrastructure that runs the server.

1.1.12 This application is designed to serve individual groups, but on what scale?

Django, the application server that our application uses has wide acceptance in the information technology industry. We believe that the software can be made to scale up to support computing at a supercomputer scale.

Our current focus, based on the priorities of the funding that we have received, was to develop a system that serves groups consisting of dozens of bioinformaticians interacting with hundreds of end-users.

Note how the limitations would occur only at the level of running simultaneous analyses. There are no constraints for the number of users that can access/read/share/copy/customize/download recipes. Hundreds of thousands of users could be browsing the recipes with millions of page views every month.

1.1.13 What are the minimum requirements for installing the web application?

The bioinformatics recipes software itself has extremely-low memory and CPU overhead. We estimate a few hundred megabytes and less than 1% CPU utilization.

Of course, when we run an analysis, the resource utilization depends on the tasks in the processes that are employed - what is important to note is that our software imposes minimal overhead.

1.1.14 Where (and how) recipes actually run when you execute them through the web application?

The recipes are currently executed on the same platform that runs the webserver. Since the service itself has minimal overhead, the entire computational infrastructure is available for computation.

The Recipes web application comes with a built-in job queuing system that can be customized for a desired amount of parallelism. Site owners can set up more or fewer simultaneous job execution strategies depending on their computational resources.

The architecture of the server is modular. We do foresee adding a job runner that integrates with job queuing systems like PBS or SLURM.

1.1.15 How are new recipes added to the system?

Recipes get created in two ways: either from *scratch* by selecting the “Create recipe” button or by “copying” or “cloning” existing recipes. We describe the processes in more detail in other answers in this FAQ.

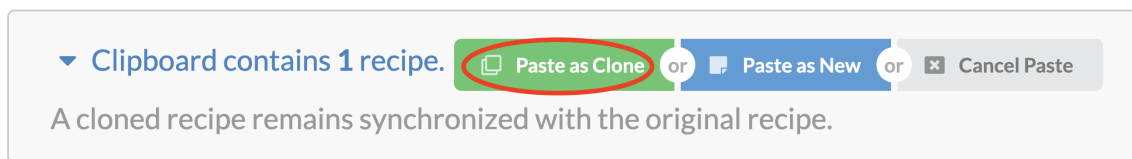
1.1.16 How do you standardise the way the recipes are created?

The problem of standardization is essential yet somewhat of a challenge to implement in a way that is not overbearing and limiting while maintaining utility to the users.

We have chosen the model of **cloning** and **copying** as described below, but we are open to suggestions from the community and may revisit the implementation later.

1.1.17 What is a “cloned” recipe?

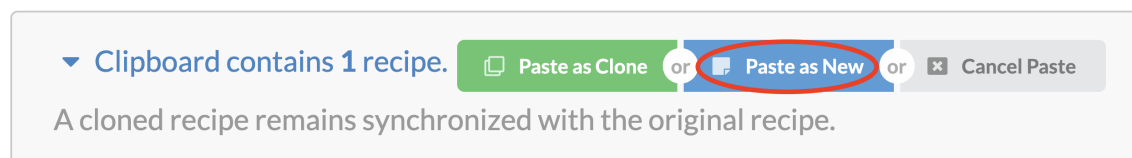
When pasting a copied recipe, we may paste it as a clone.



A cloned recipe remains in sync with the original recipe that it was cloned from. Clones cannot be changed and track the original recipe. A change to the original recipe will immediately be reflected in all the clones. The purpose of a cloned recipe is to ensure that a recipe is the same across multiple projects and individuals.

1.1.18 What is a “copied” recipe?

When pasting a copied recipe we may paste it as a new recipe.



Another method for duplicating a recipe is to copy then paste it as a new recipe. A copied recipe is a new, original recipe filled with the content from an existing recipe.

When a recipe is pasted as new, the provenance to the original recipe is not maintained. It becomes the responsibility of the author of the recipe to maintain the relevant information in the documentation of the recipe.

1.1.19 What conventions should be followed? How should they be documented? What is the minimum amount of provenance that the scripts should produce? How should that be presented to the users?

Determining the appropriate levels of documentation and provenance is a difficult question that we still debate and discuss.

We would like to avoid being either too lax or too stringent. The concepts that we popularize in this platform are new; the approach is different from past models.

We do plan to evolve our views as needed. Currently, we chose to approve only the recipes where the documentation is appropriate, and provenance is properly noted. Hence the “approved” state of a recipe is a manually curated process that indicates a higher level of standard. We hope that with time, as

CHAPTER 2

Installation

The code in **Biostar Recipes** requires Python 3.6 or above.

Our installation instructions rely on `conda` though other alternatives for managing python environments are equally viable.

```
# Create a virtual environment.
conda create -y --name engine python=3.6

# Activate the python environment.
conda activate engine

# Clone the source server code and the recipe code.
git clone https://github.com/ialbert/biostar-central.git

# Switch to the biostar-central directory.
cd biostar-central

# Install server dependencies.
pip install -r conf/requirements.txt
```

The installation is now complete. All server management commands are run through `make` by running one or more `make` tasks. For example to test the `recipes` app run:

```
make recipes test
```

2.1 Running a Demo

To run the demonstration version of the `recipes` app execute:

```
make recipes demo
```

Visit <http://127.0.0.1:8000/> to view the site.

2.2 Initialize Recipes

Activate the engine virtual environment.

```
conda activate engine
```

Migrate the recipes app by executing the command:

```
python manage.py migrate --settings biostar.recipes.settings
```

Collect static files for the recipes app by executing the command:

```
python manage.py collectstatic --noinput -v 0 --settings biostar.recipes.settings
```

There is a Makefile command that migrates and collects static files in one shot.

```
make recipes init    # Migrate and collect static files.
```

A database has now been created and the static files can be found in `biostar-central/export/static/`

To ensure installation and migration was successful, run a test by executing the command:

```
make recipes test    # Run tests.
```

To populate the database with random data run:

```
make recipes startup
```

2.3 Start Server

Activate the engine virtual environment:

```
$ conda activate engine
```

Start a local server:

```
make recipes serve    # Start local server
```

The site is now available at `http://127.0.0.1:8000/`.

When the site initializes the admin username and password are using the `ADMINS` and the `ADMIN_PASSWORD` settings in `biostar/accounts/settings.py`.

By default both the admin login name and the default admin password are set to

```
admin@localhost
```

The Django admin can be found at `http://127.0.0.1:8000/accounts/admin/`.

Customize Settings

DO NOT add your custom settings into the public codebase!

The proper practice is to create a separate, independent settings file, then, within that file import **all** default settings. Finally override the fields that you wish to customize in your settings file. For example create the `my_settings.py` then add into it:

```
# Import all default settings.  
from biostar.recipes.settings import *  
  
# Now override the settings you wish to customize.  
ADMIN_PASSWORD = "foopass"
```

Apply this settings file with

```
python manage.py runserver --settings my_settings.py
```

Consult the [Django documentation](#) for details.

3.1 Directory Structure

Each project has a physical directory associated on the system located on the system.

1. Projects directory
 - Each project has a directory with the data associated.
2. Results directory
 - Location where the results of a recipe run are stored.
3. Table of contents directory
 - Contains table of content files for every data.

These directories all found in the media directory found in the `settings.py` under `MEDIA_ROOT`. The general structure is:

```
media/  
  projects/  
    ...  
  jobs/  
    ...  
  tocs/  
    ...
```

CHAPTER 4

Deploying Site

The software follows the recommended practices for developing and deploying [Django web applications](#) .

The [Django documentation](#) contains a wealth of information on the alternative ways to deploy the site on different infrastructure.


Within this setup we recommend the `[uwsgi][uwsgi]` based deployment.


Projects


The platform is project based. Each project is a collection of data, recipes and results.


Thus each project has three distinct sections:

1. Data - the input files.
2. Recipes - the code that processes the data.
3. Results - the directory that contains the resulting files of applying the recipe to data.

 **Project**

 3 Data

 7 Recipes

 9 Results

Project Information


Owner Access


This project contains simple recipes that demonstrate the process of creating a recipe


Follow the **instructions**, investigate the **data** and **recipe code** to gain a deeper understanding of how recipes work.

Public

Project owned by [Bioinformatics Admin](#) • Updated 4 weeks ago by [Bioinformatics Admin](#)

 Edit Project

 Manage Access

 Delete Project

5.1 Project Privacy

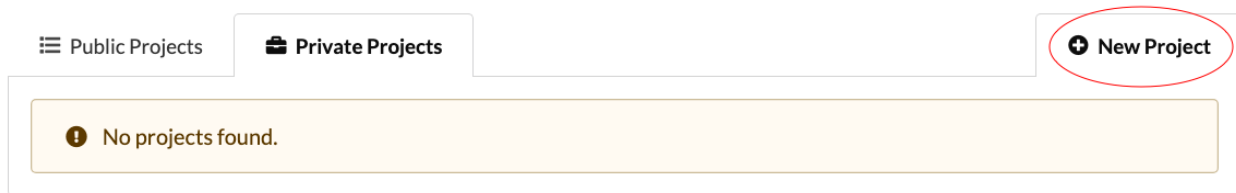
Within the management interface, all content is grouped into projects that may have public or private visibility. Content stored in public projects is readable without restrictions. Private projects will restrict access to members only.

Public Project owned by [Bioinformatics Admin](#) • Updated 4 weeks ago by [Bioinformatics Admin](#)

1. Public - viewable to everyone
2. Private - viewable to collaborators
3. Sharable - actively shared amongst a set of users

5.2 Create a Project

Click on the `New Project` tab circled on the right.



This will bring you to a form to fill in the name, privacy, information, etc...

Project Title

Project Name

What do you want to call the project

Project Image

Choose File No file chosen

Optional image to recognize the project by (square image, no more than 500px)

Project Rank

100

Used in ordering project lists on the page

Project Description

project description

A detailed explanation of the project (markdown OK).

Create

Cancel

5.3 Project Access

The web application provides a transparent and consistent framework to conduct analyses that can be shared among collaborators or with the public.

Recipes, data and results can be copied across projects, users may create new projects and may allow others (or the public) to access the contents of a project.

Access level and their respective permissions are:

Public:

- Clone and copy recipes.
- Read and copy data.
- Read and copy results.

Read:

- Clone and copy recipes.
- Read and copy data.
- Read and copy results.

- Create and edit their own recipes.
- *Trusted users* : can run recipes.

Users without read access are informed of their restrictions when trying to create a recipe.

Trusted users without read access to a recipe are also informed of their restrictions when trying to run it.


Share:

- Includes all read access permissions.
- Activated using a sharable project link

Write:

- Includes all read access permissions.
- Can upload data
- Can delete objects from project.
- Can edit all recipes in the project.
- Add or remove collaborators to the project

Users that try to edit a recipe without write access are informed of their limitations in this project with:

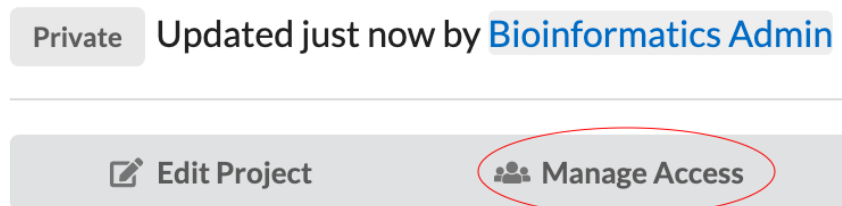
 You need write access to the original recipe to edit.

Users without write access that try to upload data or delete objects are informed of their restrictions using a message.

You need  Write Access to perform that action.


5.4 Granting Access

Click on a project and open the first tab.







Click on the middle button labeled Manage Access

Search for users using their username, uid, or name. You can select their


Search results

Filtering for : **user** • [↺ Clear](#)

 Aswathy Trusted @9c88dbc5	<div>No Access ▼</div>
 mrmuchow @c8ea3d13	<div>No Access ▼</div>
 testing @1838f658	<div>No Access ▼</div>
 Natay Aberra Trusted @8b6c9d7d	<div>No Access ▼</div>

Data may be uploaded or may be linked directly from a hard drive or from a mounted filesystem, thus avoiding copying and transferring large datasets over the web. For recipes that connect to the internet to download data, for example when downloading from the `Short Read Archive` the data does not need to be already present in the local server.

Notably the concept of “data” in our system is broader and more generic than on a typical file system. In our software “data” may be a single file, it may be a compressed archive containing several files or it may be a path to a directory that contains any number of files as well as other subdirectories. The programming interfaces for recipes can handle directories transparently and make it possible to run the same recipes that one would use for a single file on all files of an entire directory.

6.1 Data Types

Data types are labels (tags) attached to each data that help filtering them in dropdown menus. More than one data type may be listed as comma separated values. The data types may be any word (though using well recognized names: `BED`, `GFF` is recommended).



6.2 Upload Data

Data can be added multiple ways.

Web interface options:

- Upload a file
- Write text


- Import directory

Open the `Project` tab inside of a project.

Then click on the `Add Data` button

This opens another a form with two options.


- 1 . **Upload a file** - Comes with size restrictions that can be found in the `settings.py`

 **Upload a file**

Your storage space is **5.0 MB**. Across all projects you use **11.2 MB**.

No file chosen

File or zip collection of files you want to analyze.

 **Direct Data Entry**

You may manually create a dataset here.

Name

Name of the data.

Input Text

Enter data as text instead of uploading a file (10000 characters).

- 2 . **Write text** - 10k character limit

6.3 Import Directory

Admin, staff, and trusted users can see an extra tab labeled `Import Data`

7.1 Recipe Ingredients

Each recipe is built from two ingredients:

1. The interface specification file.
2. The template specification file.

The **interface** will specify the value of the parameters that get substituted into the **template**.

The **template** contains the commands that need to be executed. The **template** will have placeholders for the parameter values that the user will need to enter in the interface.

The interface + template will generate a script that the site can execute.

The software will generate an web interface for each parameter specified in the interface. It is this interface where users are able to select the values that their recipe needs to operate.

A recipe consists of a “TOML definition file” and a “script template”.

The simplest TOML definition file is an empty file and a simple script template might contain just:

```
echo 'Hello World!'
```

The **Results** are created by applying a **Recipe** on **Data**.

7.2 Create a Recipe

Creating a recipe can be done using the command line or web interface.

Web interface:

- Create a brand new recipe.
- Clone or copy one from another project.

7.2.1 Brand New Recipe

Users have the option of creating a brand new recipe or copying/cloning one from an existing one.

To create a brand new recipe, click on the `Project` tab located on the left and find the `Create Recipe` button.

Project information goes here.

Public Project owned by [Admin User](#) • updated 2 hours ago by [Admin User](#)

[Edit project](#) [Create recipe](#) [Add Data](#) [Manage access](#) [Delete project](#)

This takes you to the following page.

Run


Information

Results (3)

Code

Interface

Download



Visualize FASTQ data quality

Generates a FastQC report on a single file or a data collection. For more information see:

[3 results](#) • updated 2 hours ago by [Admin User](#)

Generates a FastQC report on a single file or a data collection. For more information see:

- [Bioinformatics Data Analysis](#) online course
- [Biostar Handbook](#) for system setup and other information.

[Copy recipe](#) [Edit description](#) [Delete recipe](#)

7.2.2 Copy or Clone

After clicking `Copy` the recipe is in your clipboard. Open the `Recipe` tab of any project to view your clipboard.

Once your clipboard has recipes, you can **clone** or **copy** them.

Cloning allows your recipes to stay up to date with an original source.

Note You can clone with `Read Access` and edit the cloned recipe with `Write Access` to the original.

A cloned recipe remains in sync with the original recipe that it was cloned from. Clones cannot be changed and track the original recipe. A change to the original recipe will immediately be reflected in all the clones. The purpose of a cloned recipe is to ensure that a recipe is the same across multiple projects and individuals.

To paste the recipes as a clone, click the `Paste as clone` at the top of the `Recipes` tab.

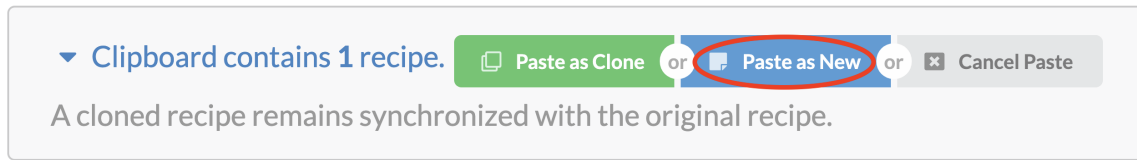
▼ Clipboard contains 1 recipe.

[Paste as Clone](#) or [Paste as New](#) or [Cancel Paste](#)

A cloned recipe remains synchronized with the original recipe.

The second method to duplicate a recipe is to copy it. A copied recipe is a brand new recipe filled with the content from an existing recipe. When a recipe is copied the provenance to the original recipe is not maintained. It becomes the responsibility of the author of the recipe to maintain the relevant information in the documentation of the recipe.

To paste the recipes as a new one, click the Paste as New at the top of the Recipes tab.



7.3 Interface Specification

The TOML definition file lists the parameters and allows the interface to be rendered. Here is an example TOML definition file:

```
[reads]
value = "FASTQ Data Collection"
label = "Sequencing Reads"
type = "FASTQ"
source = "PROJECT"

[group]
label = "Plot features"
display = "DROPDOWN"
choices = [ [ "default", "Default",], [ "nogroup", "No Grouping",],]
value = "default"
help = "Turns on/off binning in the plots."
```

Each recipe parameter will have an automatic attribute called `value` that contains either the selected value (if the parameter is user supplied) or the default value found in the interface specification file.

the parameter name is `foo`, the default value is `World!`. The `display` field specifies the type of the HTML widget, the `label` and `help` fields describe the interface. The interface generated from this specification file looks like this:

Generated Interface

7.4 Interface Builder

One of the useful features in our web interface is the **interface builder**. We found building interfaces to be the most cumbersome process in the recipes workflow so we created a feature that would build the specification file for you.

Recipe Interface Builder

Click the buttons on the right to create new fields.

Sequencing Reads:

A100_V4_10K_1_soil.fastq.gz

This should be a collection of FASTQ files. Required data type: FASTQ

Plot features:

Default

Turns on/off binning in the plots.

☒ Summarize with MultiQC
Check this to produce a single plot with all data overlaid.

✓ Run

⌂ Cancel

+ Add text field

+ Add float field

+ Add data field

+ Add checkbox

+ Add dropdown

+ Add upload field

+ Add integer field

+ Add radio button

Interface Editor

Edit the content of each interface element.

```

1 [reads]
2 value = "FASTQ Data Collection"
3 label = "Sequencing Reads"
4 type = "FASTQ"
5 display = "DROPDOWN"
6 source = "PROJECT"
7 help = "This should be a collection of FASTQ files."
8
9 [group]
10 label = "Plot features"
11 display = "DROPDOWN"
12 choices = [ [ "default", "Default",], [ "nogroup", "No Grouping",],]
13 value = "default"
14 help = "Turns on/off binning in the plots."
15
16 [summarize]
17 label = "Summarize with MultiQC"
18 display = "CHECKBOX"
19 value = true
20 help = "Check this to produce a single plot with all data overlaid."

```

7.5 Data Field

A “data” unit in the `recipes` app is a directory that may contain one or more (any number of files).

7.5.1 Data value

Each recipe parameter will have an automatic attribute called `value` that contains either the selected value (if the parameter is user supplied) or the first file from the `table-of-contents`. For data consisting of a single file one may use the value directly.

```
fastqc {{reads.value}}
```

7.5.2 Table of Contents

Each recipe parameter will have an automatically generated attribute called `toc` (table of contents) that returns the list of the file paths in the data.

The file paths are absolute paths. The `toc` can be used to automate the processing of data. For example a data directory named `reads` contains several FASTQ files with `.fq` extensions. To run `fastqc` on each file that matches that the recipe may use:

```
cat {{reads.toc}} | grep .fq | parallel fastqc {}
```

7.5.3 Data Source

When a recipe parameter indicates the source of the parameter as `PROJECT` it will be populated from the data in the project that matches the type.

```
[reads]
value = "FASTQ Data Collection"
label = "Sequencing Reads"
type = "FASTQ"
source = "PROJECT"
```

Only data that matches the tag `FASTQ` will be shown in the dropdown menu.

7.5.4 Data Types

Data types are labels (tags) attached to each data that help filtering them in dropdown menus. More than one data type may be listed as comma separated values. The data types may be any word (though using well recognized names: `BED`, `GFF` is recommended).

Data that exists on a filesystem may be linked into the Biostar Engine from the command line. This means that no copying/moving of data is required. The only limitation is that of the filesystem.

7.6 Recipe Template

A recipe is a script that has template markers for filling in parameters. In the case for the `foo` variable above, we can access its value via:

```
echo 'Hello {{foo.value}}'
```

Recipes are using [Django templates](#) and may contain Django template specific constructs.

When the recipe is run the template will be substituted according to the interface value entered by the user. If the default value is kept it will produce the script:

```
echo 'Hello World!'
```

7.7 Recipe Execution

Before executing the recipe the script template is rendered with the JSON data and is filled into the template.

```
template + TOML -> script
```

The script is then executed at the command line.

The recipe execution creates a `Result` objects.

7.8 Job Runner

The platform users an asynchronous task scheduler to execute the recipes in the background. The site admins has control on how many workers are spawned and how many are used to run recipes.

Results consists of all files and all the metadata created by the recipe as it is executed on the input data.

Each run of a recipe will generate a *new* result directory. Users may inspect, investigate and download any of the files generated during the recipe run. Additionally, users may copy a result file as new data input for another recipe.

8.1 Output Directory

Once the recipe runs a results directory is created that contains the following:

- the code for the recipe
- the standard out and error stream content
- all files created by the recipe

The results directory is a snapshot of all files generated when the recipe has been run, including the recipe itself.

8.2 Rerun Results

Site administrator with shell access to the server can use these commands to interact with the recipes platform in an automated fashion.

9.1 Create a Project

Use the management command `project` to create a project from command line.

```
$ python manage.py project --help

usage: manage.py project [-h] --pid PID [--name NAME] [--info INFO] [--public]
                        [--update] [--version] [-v {0,1,2,3}]
                        [--settings SETTINGS] [--pythonpath PYTHONPATH]
                        [--traceback] [--no-color] [--force-color]

Creates a project.

optional arguments:
  -h, --help            show this help message and exit
  --pid PID             Project id
  --name NAME           Project name
  --info INFO           File path or text of the project info
  --public              Makes project public
  --update              Updates the project selected by pid
  ...
```

Note: The owner of any project created from command line is an first admin user.

To create a sample project, run the command:

```
python manage.py project --name sample project --public --info "This is a sample" --
↪pid sample
```

9.2 Granting Access

Adding collaborators can be done using the command line or the interface.

To add a user using command line use the management command `add_user`:

```
$ python manage.py add_user --help --fname user_file.csv

usage: manage.py add_user [-h] [--fname FNAME] [--version] [-v {0,1,2,3}]
                        [--settings SETTINGS] [--pythonpath PYTHONPATH]
                        [--traceback] [--no-color] [--force-color]

Add users

optional arguments:
  -h, --help            show this help message and exit
  --fname FNAME          The CSV file with the users to be added. Must have
                        headers: Name, Email
```

With a sample csv file `user_list.csv` that looks like :

```
user 1,  user1@email
user 2,  user2@email
```

You can run the following command using the file:

```
python manage.py add_user --fname user_list.csv
```

9.3 Upload Data

Command line options:

- Link a file directly from a hard drive

You can use the management command `data` to add or edit Data objects.

```
$ python manage.py data --help

usage: manage.py data [-h] --pid PID [--did DID] [--update] [--path PATH]
                    [--text TEXT] [--name NAME] [--type TYPE] [--version]
                    [-v {0,1,2,3}] [--settings SETTINGS]
                    [--pythonpath PYTHONPATH] [--traceback] [--no-color]
                    [--force-color]

Adds data to a project

optional arguments:
  -h, --help            show this help message and exit
  --pid PID              Select project by unique uid
  --did DID              Select data by unique uid
  --update               Update the table of content for data --did.
  --path PATH            Path to the data to be added (file or directory)
  --text TEXT            A file containing the description of the data
  --name NAME            Sets the name of the data
  --type TYPE            Sets the type of the data
```


Link a sample directory, /path/to/data/, to an existing project with the uid project_one:

```
$ python manage.py data --pid project_one --path /path/to/data/ --name New data
```

9.4 Create a Recipe

Creating a recipe can be by directly upload json and script template to a given recipe.

Use the recipe management command to directly add to a project.

```
$ python manage.py recipe --help

usage: manage.py recipe [-h] --pid PID --rid RID [--json JSON]
                        [--template TEMPLATE] [--info INFO] [--name NAME]
                        [--image IMAGE] [--update] [--version] [-v {0,1,2,3}]
                        [--settings SETTINGS] [--pythonpath PYTHONPATH]
                        [--traceback] [--no-color] [--force-color]

Adds recipe to a project

optional arguments:
  -h, --help            show this help message and exit
  --pid PID             Project id.
  --rid RID             Recipe id.
  --json JSON           Recipe json path.
  --template TEMPLATE  Recipe template path (optional)
  --info INFO           Recipe description (optional)
  --name NAME           Recipe name
  --image IMAGE         Recipe image path
  --update             Updates the recipe
```

For example, the command below would add a recipe named New recipe to project with uid 1.

```
python manage.py recipe --pid 1 --name New recipe --json < interface file > --
↪template < script template >
```


10.1 Commands

Pull API:

```
$ python manage.py api pull --help

optional arguments:
  -h, --help            show this help message and exit
  -r, --recipes          Pull recipes of --pid
  --url URL              Site url.
  --key KEY              API key. Required to access private projects.
  --rid RID              Recipe uid to dump.
  --pid PID              Project uid to dump.
  --dir DIR              Directory to store in.

# Dump project from remote url (--url).
$ python manage.py api pull --pid tutorial --dir tmp/remote/projects/ --url URL

# Dump recipes from remote url ( --url)
$ python manage.py api pull --pid tutorial --dir tmp/remote/recipes/ --url URL --
↪ recipes
```

Push API:

```
$ python manage.py api push --help

optional arguments:
  -h, --help            show this help message and exit
  -u, --url_from_json    Extract url from conf file instead of --url.
  --url URL              Site url.
  --key KEY              API key. Required to access private projects.
  --rid RID              Recipe uid to load.
  --pid PID              Project uid to load.
  --dir DIR              Directory with json files to load in bulk.
```

(continues on next page)

(continued from previous page)

```
--json JSON          Project or recipe JSON file to load.

# Load project tutorial from json file.
$ python manage.py api push --json ../biostar-recipes/projects/tutorial.hjson

# Load recipe jsons in --dir to project --pid. Upload to remote host with -u flag.
$ python manage.py api push --pid tutorial --dir ../biostar-recipes/recipes/ -u --
↪key API_KEY
```

10.2 Methods

10.2.1 Listing

```
GET /api/list/
```

List projects and recipes in a tab delimited fashion with columns: **Project ID** , **Project Name**, **Recipe ID**, **Recipe Name**, **Privacy**

Example

```
/api/list/
```

tutorial	Recipe Tutorials	environment	Environment	
↪Check	Public			
tutorial	Recipe Tutorials	interface	Interface	
↪Elements	Public			
tutorial	Recipe Tutorials	makefile	Makefile Example	Public
tutorial	Recipe Tutorials	rscript	R Script	Public
cookbook	Bioinformatics Cookbook	quality-check	Improve the	
↪quality of sequencing reads	Public			
cookbook	Bioinformatics Cookbook	fastqc	Visualize FASTQ data	
↪quality	Public			
cookbook	Bioinformatics Cookbook	pseudo-alignment	RNA-Seq	
↪Transcript Abundance Estimation	Public			
cookbook	Bioinformatics Cookbook	augustus	Gene	
↪Prediction	Public			

10.2.2 Project Information

```
GET /api/project/{id}/
PUT /api/project/{id}/
```

Parameters

- *id*: unique project ID

Example

/api/project/tutorials/

```
{
  settings:
  {
    uid: tutorial
    name: Recipe Tutorials
    image: tutorial.png
    privacy: Public
    help:
      '''
      This project contains simple analyses that demonstrate the process
      of creating a recipe.

      Follow the instructions, investigate the data and recipe code
      to gain a deeper understanding of how recipes work.

      Read the step by step instructions in the [How to write recipes] (https://github.com/biostars/biostar-recipes/blob/master/docs/how-to-write-recipes.md).
      '''
    id: 2
    project_uid: tutorial
    url: http://localhost:8000
  }
  recipes:
  [
    empty
    makefile
    starter
    interface
    hello-world
    environment
    rscript
  ]
}
```

10.2.3 Project Image

```
GET /api/project/image/{id}/
PUT /api/project/image/{id}/
```

Parameters

- *id*: unique project ID

Example

/api/project/image/tutorials/

Image in response:



center

20%

10.2.4 Recipe Json

```
GET /api/recipe/json/{id}/  
PUT /api/recipe/json/{id}/
```

Recipe JSON used to generate interface

Parameters

- *id*: Unique recipe ID

Fields in response

Fields associated with the recipe JSON

Example

/api/recipe/json/starter/

```
{
  readlen:
  {
    label: Read Length
    display: INTEGER
    value: 250
    range:
    [
      70
      100000
    ]
  }
  instrument:
  {
    label: Select Instrument
    display: DROPDOWN
    choices:
    [
      [
        hiseq
        Illumina Hiseq
      ]
      [
        pacbio
        Pacific BioSciences Sequel
      ]
      [
        minion
        Oxford Nanopor Minion
      ]
    ]
    value: pacbio
  }
  reference:
  {
    label: Reference Genome
    display: DROPDOWN
    type: FASTA
    source: PROJECT
    value: Genome.fa
  }
  settings:
  {
    name: Starter Recipe
    image: starter.png
    help:
    """
    This recipe can be a starting point for other recipes.
    # Help
  """
  }
}
```

(continues on next page)

(continued from previous page)

```
    Use this recipe to create new recipes.
    '''
    id: 12
    uid: starter
    project_uid: tutorial
    url: http://localhost:8000
  }
}
```

10.2.5 Recipe Template

```
GET /api/recipe/template/{id}/
PUT /api/recipe/template/{id}/
```

Recipe template executed during analysis.

Parameters

- *id*: Unique recipe ID

Example

/api/recipe/template/starter/

```
#
# A starter recipe with examples.
#
#
# You can fill in shell variables
#
READLEN=250

echo "Read length: $READLEN"

#
# Substitute into content
#
echo "Referene genome: Genome.fa"

#
# But you may also use Django Template constructs.
#

echo "Yes, it is Pacific Biosciences!"

#
# Generate a table of content with all files in the job directory.
#
find . -name '*' > files.txt
```

(continues on next page)

(continued from previous page)

```
#  
# Print the contents to the screen  
#  
echo "***** File List: files.txt *****"  
cat files.txt  
# Make a nested directory  
mkdir -p foo/bar  
find . -name '*' > foo/bar/all.txt
```

10.2.6 Recipe Image

```
GET /api/recipe/image/{id}/  
PUT /api/recipe/image/{id}/
```

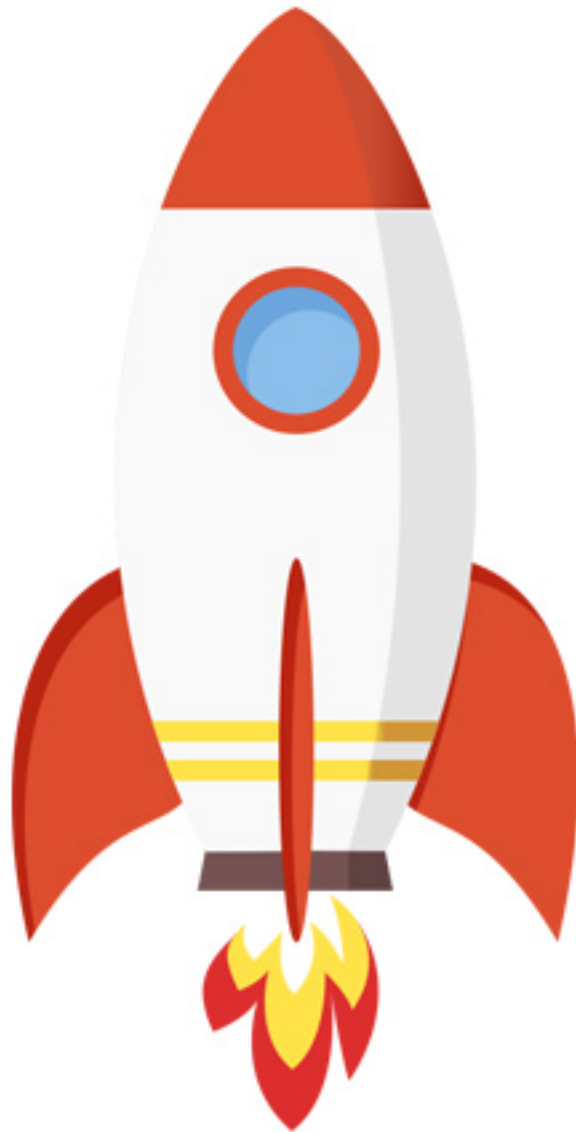
Parameters

- *id*: Unique recipe ID

Example

`api/recipe/image/starter/`

Image in response:



10.2.7 Data Update

Adding data api docs.

```
curl --form "@file=/Users/natay/Desktop/apps/biostar-central/  
SimpleWorkflowMNIST.ipynb" http://localhost:8000/api/data/data-1/
```