
Portia Documentation

Release 2.0.8

Scrapinghub

Oct 17, 2018

Contents

1	Installation	3
1.1	Docker (recommended)	3
1.2	Vagrant	4
1.3	Ubuntu	4
1.4	Developing Portia using Docker	4
2	Getting Started	7
2.1	Creating a spider	7
2.2	Creating a sample	8
2.3	Configuring your crawler	9
2.4	What's next?	10
3	Examples	11
3.1	Crawling paginated listings	11
3.2	Selecting elements with CSS and XPath	13
3.3	Extracting a single attribute to multiple fields	13
3.4	Scraping multiple items from a single page	14
3.5	Using Multiple Samples to Deal with Different Layouts	14
4	Projects	17
4.1	Versioning	17
4.2	Deployment	18
5	Spiders	19
5.1	Spider properties	19
5.2	Start pages and link crawling	20
5.3	Running a spider	21
5.4	Minimum items threshold	21
6	Samples	23
6.1	What are samples?	23
6.2	What are annotations?	23
6.3	Annotations	23
6.4	Multiple samples	26
7	Items	27
7.1	Field types	27

8	FAQ	29
8.1	How do I use Crawlera with Portia?	29
8.2	Does Portia support AJAX based websites?	29
8.3	Does Portia work with large JavaScript frameworks like Ember?	29
8.4	Does Portia support sites that require you to log in?	29
8.5	Does Portia support content behind search forms?	29
9	Indices and tables	31

Contents:

CHAPTER 1

Installation

1.1 Docker (recommended)

If you are on a Linux machine you will need [Docker](#) installed or if you are using a [Windows](#) or [Mac OS X](#) machine you will need [boot2docker](#).

You can run Portia with the command below:

```
docker run -i -t --rm -v <PROJECTS_FOLDER>:/app/data/projects:rw -p 9001:9001_
↳scrapinghub/portia
```

Or with docker-compose by running:

```
docker compose up
```

Portia will now be running on port 9001 and you can access it at <http://localhost:9001>. Projects will be stored in the project folder that you mount to docker.

To extract data using portia you can run your spider with:

```
docker run -i -t --rm -v <PROJECTS_FOLDER>:/app/data/projects:rw -v <OUTPUT_FOLDER>:/
↳mnt:rw -p 9001:9001 scrapinghub/portia \
    portiacrawl /app/data/projects/PROJECT_NAME SPIDER_NAME -o /mnt/SPIDER_NAME.jl
```

After the crawl finishes you will find your extracted data in the the *OUTPUT_FOLDER*

Note: *<PROJECT_FOLDER>* and *<OUTPUT_FOLDER>* are just paths on your system where your projects and extracted data are stored.

Warning: For Windows the *<PROJECT_FOLDER>* path must be of the form */<DRIVE_LETTER/<PATH>*. For example */C/Users/UserName/Documents/PortiaProjects*

1.2 Vagrant

Checkout the repository:

```
$ git clone https://github.com/scrapinghub/portia
```

You will need [Vagrant](#) , [VirtualBox](#) [Node.js](#), [Bower](#) and [ember-cli](#) installed.

Run the following in Portia's directory:

```
docker/compile-assets.sh
vagrant up
```

This will launch an Ubuntu virtual machine, build Portia and start the portia server. You'll then be able to access Portia at `http://localhost:9001`. You can stop the portia server using `vagrant suspend` or `vagrant halt`. To run portiacrawl you will need to SSH into the virtual machine by running `vagrant ssh`.

1.3 Ubuntu

1.3.1 Running Portia Locally

These instructions are only valid for an Ubuntu based OS

Install the following dependencies:

```
sudo ./provision.sh install_deps
```

If you would like to run Portia locally you should create an environment with virtualenv:

```
virtualenv YOUR_ENV_NAME --no-site-packages
source YOUR_ENV_NAME/bin/activate
cd ENV_NAME
```

Now clone this repository into that env:

```
git clone https://github.com/scrapinghub/portia.git
cd portia
```

Install splash and the required packages:

```
sudo ./provision.sh install_deps install_splash install_python_deps
```

To run Portia start slyd and portia_server:

```
PYTHONPATH='/vagrant/portia_server:/vagrant/slyd:/vagrant/slybot'
slyd/bin/slyd -p 9002 -r portiaui/dist &
portia_server/manage.py runserver
```

Portia should now be running on port 9001 and you can access it at `http://localhost:9001`.

1.4 Developing Portia using Docker

To develop Portia using docker you will need [Node.js](#), [Bower](#) and [ember-cli](#) installed.

To set up Portia for development use the commands below:

```
mkdir ~/data
git clone git@github.com:scrapinghub/portia.git
cd portia/portiaui
npm install && bower install
cd node_modules/ember-cli && npm install && cd ../../
ember build
cd ..
docker build . -t portia
```

You can run it using:

```
docker run -i -t --rm -p 9001:9001 \
  -v ~/data:/app/data/projects:rw \
  -v ~/portia/portiaui/dist:/app/portiaui/dist \
  -v ~/portia/slyd:/app/slyd \
  -v ~/portia/portia_server:/app/portia_server \
  portia
```

This sets up the `portia_server` to restart with every change you make and if you run `cd ~/portia/portiaui && ember build -w` in another shell you can rebuild the Portia assets with every change too.

CHAPTER 2

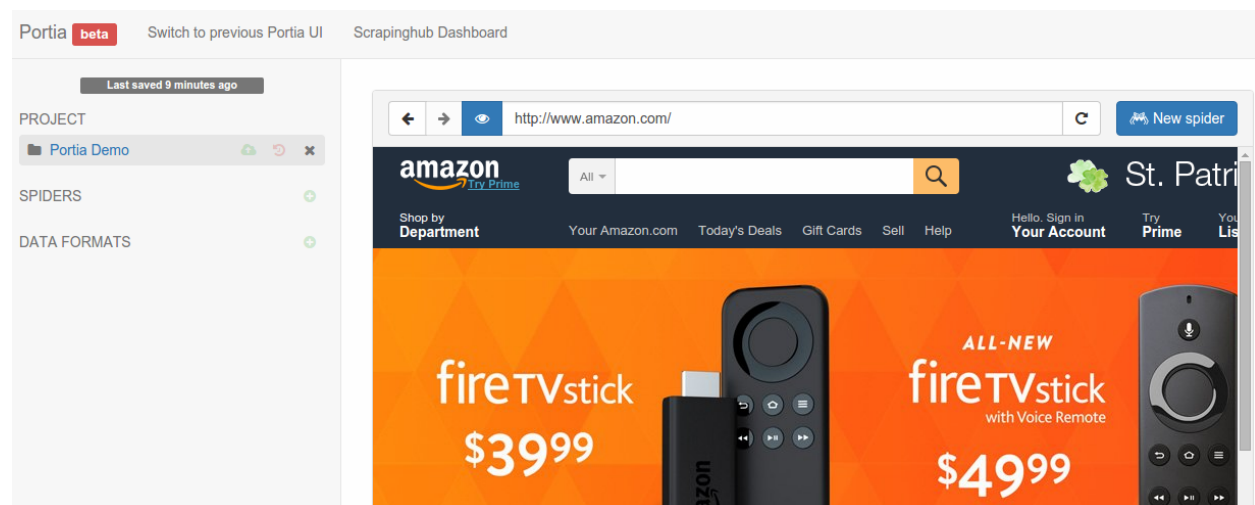
Getting Started

Note: If you don't have Portia running yet, please read the [Installation guide](#) first. If you're using a hosted version of Portia on a platform like [Scrapinghub](#), you don't need to install anything.

This tutorial will briefly cover how to begin extracting data with Portia.

2.1 Creating a spider

Let's start by creating a project. Enter a URL and Portia will render it like below:

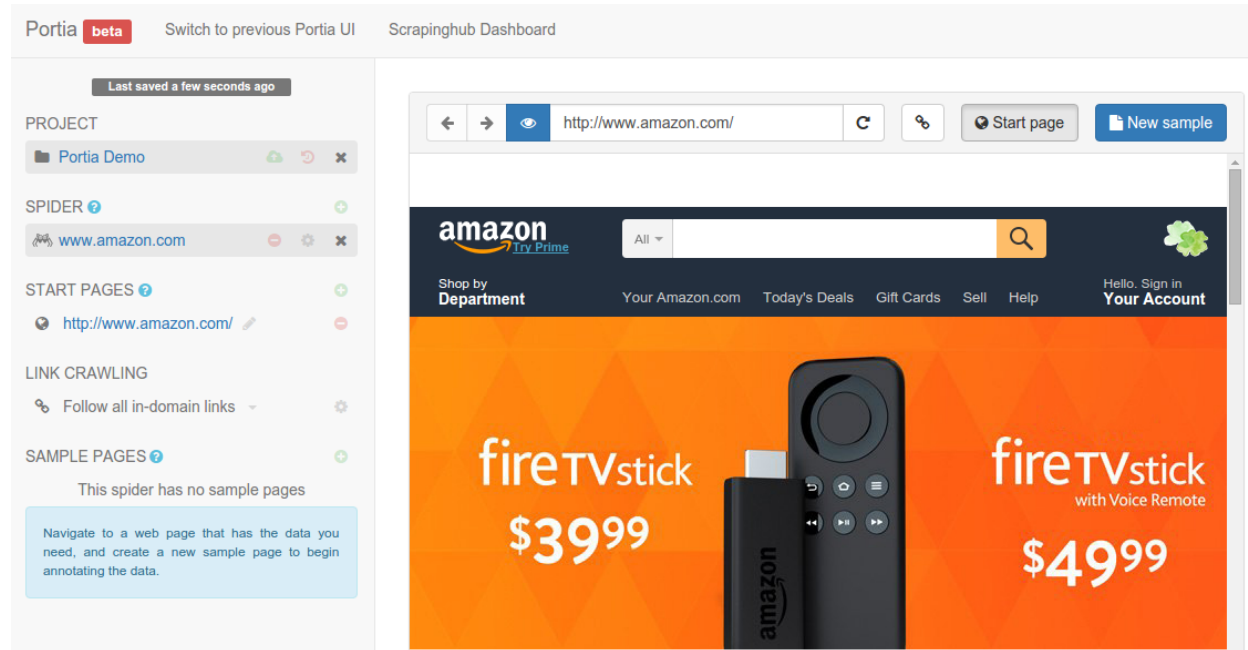


Click the `New spider` button to create a new spider. Portia will add the page's URL as a start page automatically. Start pages are used to seed the crawl and Portia will visit them when you start the spider to find more links.

2.2 Creating a sample

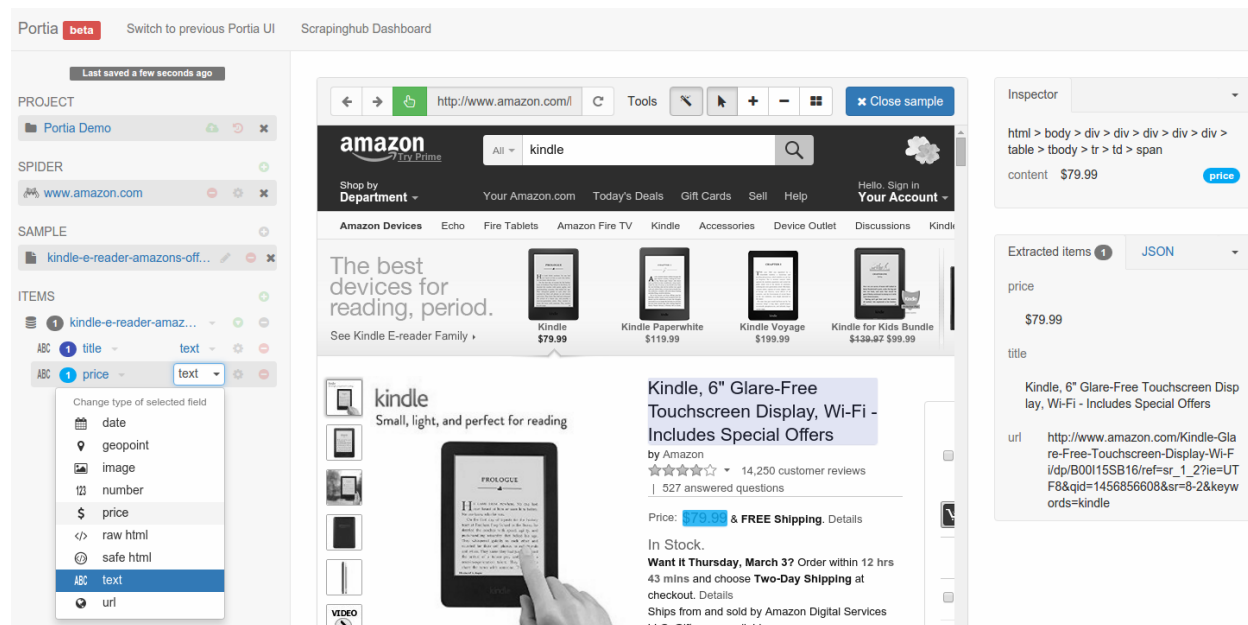
A sample describes how data should be extracted from the page. Portia will use your samples to extract data from other pages with a similar structure.

Portia works like a web browser, so you can navigate between pages as you would normally. Navigate to a page you want to scrape and then the `New sample` button to create a *sample* of the page.



Now that you've created the sample, you can begin *annotating* the page. Annotations link a piece of data in the page to an item field. You'll notice that you can highlight elements on the page, if you click on it will create a new field to which the element will be extracted.

Portia will create an *item* schema from the elements that you annotated and will use it as the data format for the scraped *items*.



You can see a preview of the items your sample will extract on the right. Once you've annotated all the data you wish to extract, close the sample. Your spider is *ready to run*, but you may want to configure it further in which case you should continue reading.

2.3 Configuring your crawler

To start crawling a website, Portia needs one or more URLs to visit first so it can gather further links to crawl. You can define these URLs on the left under `START PAGES`.

The screenshot shows the Portia web interface. At the top, there's a header with 'Portia beta', a link to 'Switch to previous Portia UI', and a link to 'Scrapinghub Dashboard'. Below the header, a status bar indicates 'Last saved a few seconds ago'. The main interface is divided into two columns. The left column contains configuration sections: 'PROJECT' with a folder icon and 'Portia Demo'; 'SPIDER' with a spider icon and 'www.amazon.com'; 'START PAGES' with two URL entries, the second of which is 'http://www.amazon.com/gp/goldbox/' and is currently selected; 'LINK CRAWLING' with a checkbox for 'Follow all in-domain links'; and 'SAMPLES' with a document icon and 'kindle-e-reader-amazons-officia...'. The right column shows a preview of the Amazon website, displaying the Amazon logo, a 'Shop by Department' dropdown, 'Amazon Devices', and a promotional banner for Kindle e-readers.

Portia follows all in-domain URLs by default. In many cases you'll want to limit the pages Portia will visit so requests aren't wasted on irrelevant pages.

To do this, you can set follow and exclude patterns that whitelist and blacklist URLs respectively. These can be configured by changing the crawling policy to `Configure URL patterns`.

For example, Amazon products' URLs contain `/gp/`, so you can add this as a follow pattern and Portia will know to only follow such URLs.

The screenshot displays the Portia beta web interface. At the top, there's a header with 'Portia beta', 'Switch to previous Portia UI', and 'Scrapinghub Dashboard'. Below this, a status bar indicates 'Last saved a few seconds ago'. The main interface is divided into several sections: 'PROJECT' with a 'Portia Demo' folder; 'SPIDER' with a configuration for 'www.amazon.com'; 'START PAGES' with two Amazon URLs; and 'LINK CRAWLING' with a dropdown menu. The dropdown menu is open, showing options: 'Change how links are crawled', 'Follow all in-domain links', 'Don't follow links', and 'Configure url patterns' (which is highlighted). To the right, a 'Link crawling options' panel is open, showing 'CRAWLING RULES'. It has two sections: 'Follow links that match these patterns' with a text input containing '/gp/' and a plus icon, and 'Exclude links that match these patterns' with a text input containing 'Regular expression' and a plus icon. At the bottom of this panel, there is a checked checkbox labeled 'Respect the "nofollow" attribute'.

2.4 What's next?

Once you've created your samples and configured crawling behavior, it's time to [run](#) your spider.

Check out the [Examples](#) to learn a few tips to be more productive with Portia.

3.1 Crawling paginated listings

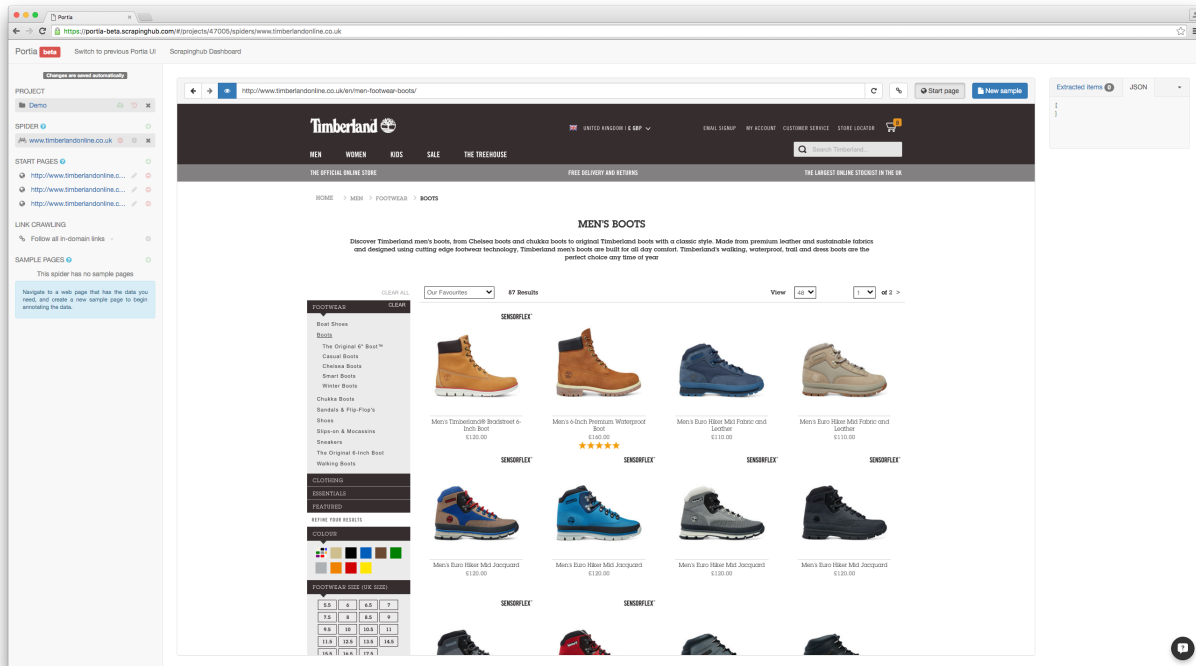
Most e-commerce sites use pagination to spread results across multiple pages.

When crawling these sites with Portia, there are some best practices you should follow:

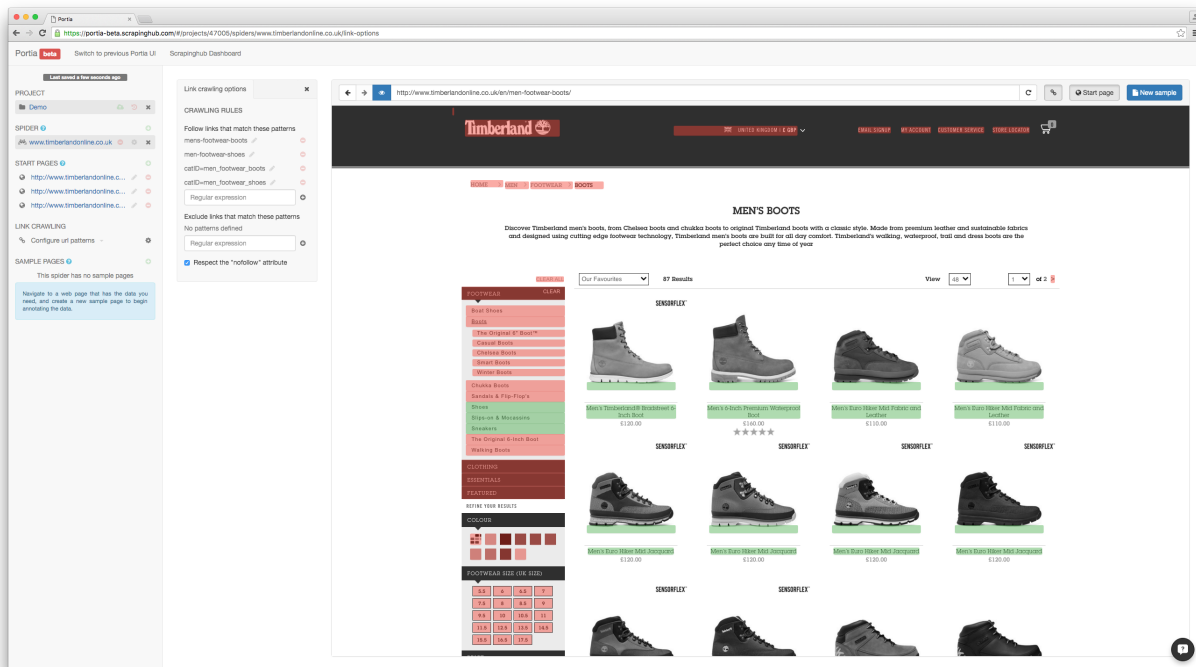
- Use the target categories as start pages.
- Use URL follow patterns to limit Portia to only visit category and article pages.


This will prevent Portia from visiting unnecessary pages so you can crawl the items a lot faster.

Let's use timberlandonline.co.uk as an example. Say you want to only scrape products from the `boots` and `shoes` categories. You can create a `spider` and add the categories to its start URLs:



To ensure the spider only visits relevant pages, you'll need to limit crawling to the target categories and product pages. You can accomplish this defining URL follow patterns in the Link Crawling configuration of your spider:



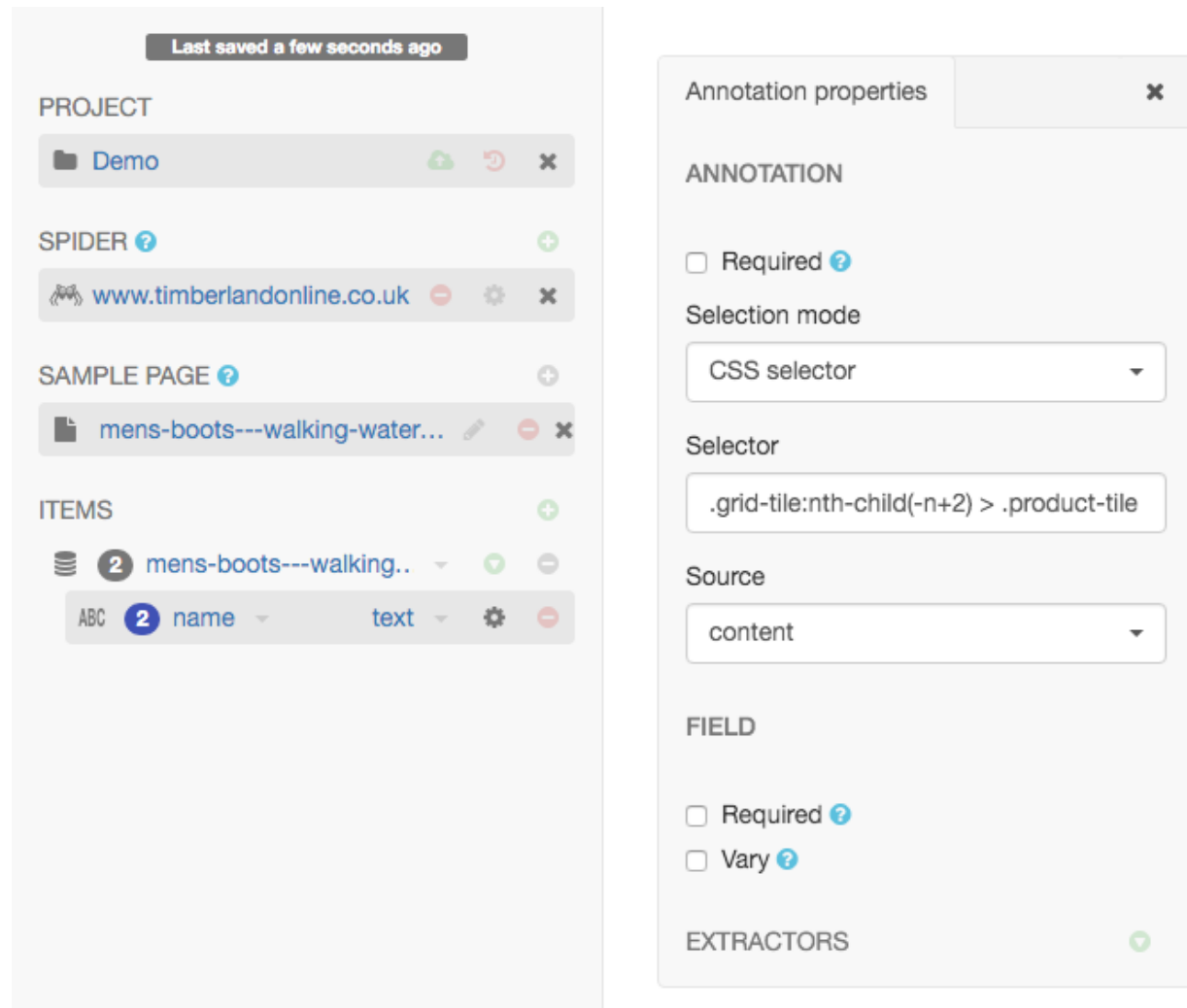
You can use follow patterns to filter URLs with **regular expressions**. You can see which links will be followed by clicking the  button (toggle highlighting) to the right of Portia's URL bar. Followed links will be highlighted in green and excluded links in red.

As you can see above, the spider will now only visit the boots and shoes category pages and their product listings. To ensure that only products belonging to the target categories are visited, we filter against the `catID` parameter value in the URL.

Crawling listings in this manner is much more efficient. You avoid visiting tons of unwanted pages on the site and instead crawl only those you need.

3.2 Selecting elements with CSS and XPath

You can select elements with CSS or XPath by changing the selection mode of an annotation. You can do it clicking the symbol right to the annotation name in the `ITEMS` section of the left sidebar.



This way, you can tweak your selections, making them more or less specific, for example.

3.3 Extracting a single attribute to multiple fields

Portia supports multiple annotations for the same attribute. You can take advantage of this to extract an attribute to multiple fields by simply creating an annotation for each field.

Imagine you want to extract the username and the date from blog posts and this information is represented like this:

```
<div class="details">
  By johndoe on March 3th
</div>
```

To extract this information separately, you have to annotate the element, click the gear icon right after the field name and add an extractor with a regular expression that captures only the username: `By (\w+) . *`.

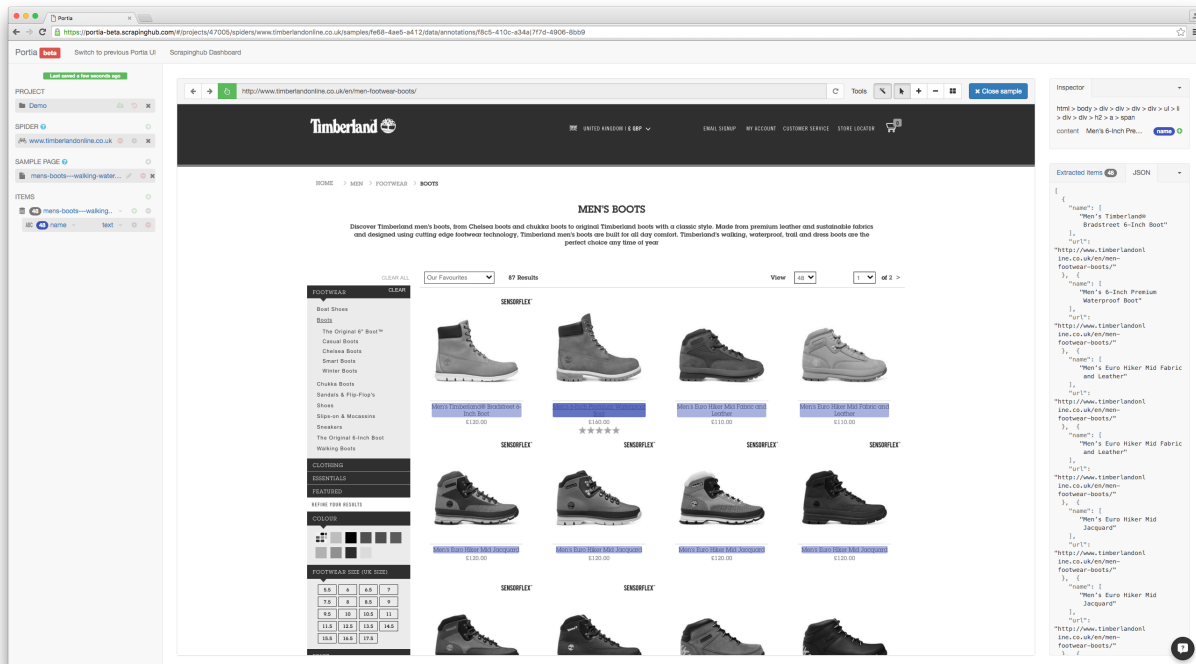
After that, you have to go back to annotation mode, click the **+** button in the toolbar and then annotate the same element again. Now, you have to create another extractor to capture only the date from the element: `By \w+ on (.*)`.

3.4 Scraping multiple items from a single page

You'll often need to retrieve several items from a single page. You can do this using either the repeating element tool **■** or with the wand **🪄** by annotating the first item's element and then clicking the second item's element. Portia will detect all similar items on the page and create annotations for each of them.

Let's revisit the timberberlandonline.co.uk spider and demonstrate this process by annotating a couple of pairs of shoes.

Click the tiles icon to select the repeating element tool and then click an element, and Portia will find all similar elements and link them to the same field:



Now you just need to do same for the other fields, and you're done!

3.5 Using Multiple Samples to Deal with Different Layouts

Some websites use different layouts to display the same kind of information. E-commerce websites usually create special pages for some products on Black Friday, for example. Sometimes, the problem is that some pages might not have all the data you need.

You can create multiple samples, even if you are extracting only one item type, to make sure your spider can handle these variations.

Consider this example: our spider has an item type with the fields `name`, `price`, `description` and `manufacturer`, where `name` and `price` are required fields. We have created a sample with annotations for each of those fields. Upon running the spider, many items are correctly scraped; however, there are a large number of scraped items where the `manufacturer` field contains what should be the `description`, and the `description` field is empty. This has been caused by some pages having a different layout:

Layout A:

```
<table>
  <tbody>
    <tr>
      <td>name</td>
      <td>price</td>
    </tr>
    <tr>
      <td colspan="2">manufacturer</td>
    </tr>
    <tr>
      <td colspan="2">description</td>
    </tr>
  </tbody>
</table>
```

Layout B:

```
<table>
  <tbody>
    <tr>
      <td>name</td>
      <td>price</td>
    </tr>
    <tr>
      <td colspan="2">description</td>
    </tr>
  </tbody>
</table>
```

As you can see, the problem lies with the fact that in layout B the `description` is where `manufacturer` would be, and with `description` not being a required field it means that the sample created for layout A will match layout B. Creating a new sample for layout B won't be enough to fix the problem, as layout A's sample *would contain more annotation and be matched against first*.

Instead we need to modify layout A's sample, and mark the `description` annotation as **Required**. With this added constraint, items displayed with layout B will not be matched against layout A's sample due to the missing `description` field, so the spider will proceed onto layout B's sample which will extract the data successfully.

[Click here to learn more about Multiple Samples.](#)

A project in Portia consists of one or more *spiders* and can be deployed to any *scrapyd* instance.

4.1 Versioning

Portia provides project versioning via Git, but this isn't enabled by default.

Git versioning can be enabled by creating a *local_settings.py* file in the *slyd/slyd* directory and adding the following:

```
import os

SPEC_FACTORY = {
    'PROJECT_SPEC': 'slyd.gitstorage.projectspec.ProjectSpec',
    'PROJECT_MANAGER': 'slyd.gitstorage.projects.ProjectsManager',
    'PARAMS': {
        'storage_backend': 'dulwich.repo.Repo',
        'location': os.environ.get('PORTIA_DATA_DIR', SPEC_DATA_DIR)
    },
    'CAPABILITIES': {
        'version_control': True,
        'create_projects': True,
        'delete_projects': True,
        'rename_projects': True
    }
}
```

You can also use MySQL to store your project files in combination with Git:

```
import os

SPEC_FACTORY = {
    'PROJECT_SPEC': 'slyd.gitstorage.projectspec.ProjectSpec',
    'PROJECT_MANAGER': 'slyd.gitstorage.projects.ProjectsManager',
```

(continues on next page)

(continued from previous page)

```
'PARAMS': {
    'storage_backend': 'slyd.gitstorage.repo.MysqlRepo',
    'location': os.environ.get('DB_URL'),
},
'CAPABILITIES': {
    'version_control': True,
    'create_projects': True,
    'delete_projects': True,
    'rename_projects': True
}
}
```

This will store versioned projects as blobs within the MySQL database that you specify by setting the environment variable below:

```
DB_URL = mysql://<USERNAME>:<PASSWORD>@<HOST>:<PORT>/<DB>
```

When this env variable is set the database can be initialized by running the `bin/init_mysqldb` script.

Note: The MySQL backend only stores project data. Data generated during crawl is still stored locally.

4.2 Deployment

You can deploy your Portia projects using `scrapy`. Change directory into `slyd/data/projects/PROJECT_NAME` and add your target to `scrapy.cfg`. You'll then be able to run `scrapy-deploy` which will deploy your project using the default deploy target. Alternatively, you can specify a target and project using the following:

```
scrapy-deploy your_scrapy_target -p project_name
```

Once your spider is deployed, you can schedule your spider via `schedule.json`:

```
curl http://your_scrapy_host:6800/schedule.json -d project=your_project_name -d_
↳ spider=your_spider_name
```

Warning: Running scrapy from your project directory will cause deployment to fail.

Spiders are web crawlers that use *samples* to extract data from the pages it visits.

5.1 Spider properties

You can access your spider's properties by clicking the gear icon located right of your spider in the list on the left.

Spider properties

INITIALIZATION

☐ Perform login

JAVASCRIPT SUPPORT

☐ Enable Javascript

5.1.1 Configuring login details

If you need to log into a site, you can configure login details by ticking ‘Perform login’ in the *spider properties* menu. Here you can set the login URL, username and password.

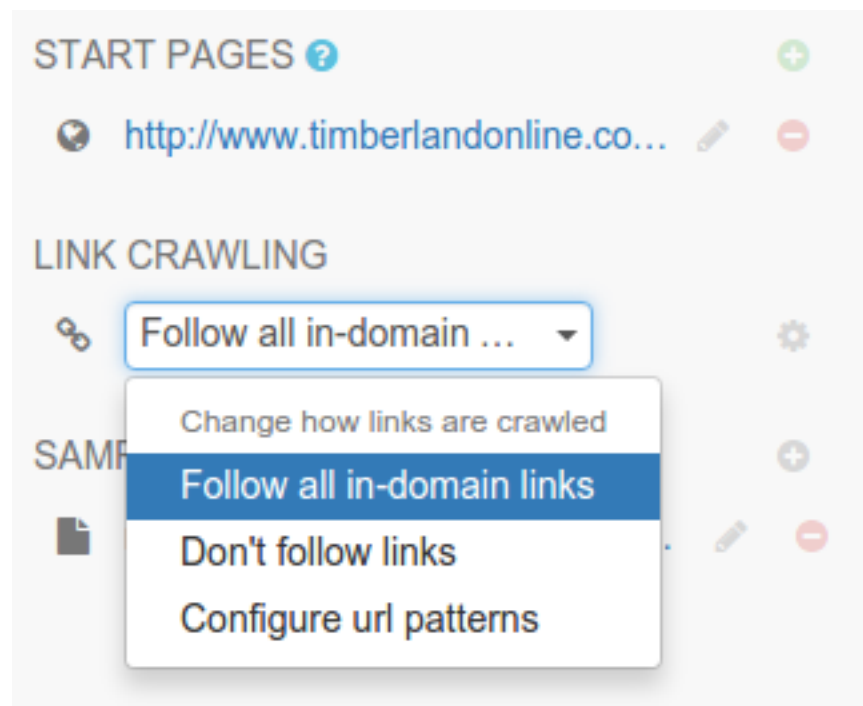
5.1.2 Enabling JavaScript

You can enable JavaScript in your spider by ticking `Enable JavaScript` in the *spider properties* menu. Note that you’ll need to set the `SPLASH_URL` Scrapy setting to your [Splash](<https://github.com/scrapinghub/splash>) endpoint URL for JavaScript to work during the crawl.

5.2 Start pages and link crawling

Start pages are the initial URLs that Portia will visit to start the crawl. You can add and remove start pages on the left menu.

You can choose how Portia will follow links under `LINK CRAWLING`.



- Follow all in-domain links - follow all links under the same domain and subdomain.
- Don't follow links - only visit start URLs.
- Configure url patterns - use regular expressions to choose which URLs to follow.

The `Configure url patterns` option lets you set follow and exclude patterns as well as choose whether to respect the `nofollow` attribute. Click the gear icon to show the link crawling options where you can set the follow/exclude patterns.

5.3 Running a spider

Portia will save your projects in `slyd/data/projects`. You can use `portiacrawl` to run a spider:

```
portiacrawl PROJECT_PATH SPIDER_NAME
```

where `PROJECT_PATH` is the path of the project and `SPIDER_NAME` is a spider that exists within that project. You can list the spiders for a project with the following:

```
portiacrawl PROJECT_PATH
```

Portia spiders are ultimately [Scrapy](#) spiders. You can pass Scrapy arguments when running with `portiacrawl` using the `-a` option. You can also specify a custom settings module using the `--settings` option. The [Scrapy documentation](#) contains full details on available options and settings.

5.4 Minimum items threshold

To avoid infinite crawling loops, Portia spiders check to see if the number of scraped items meet a minimum threshold over a given period of time. If not, the job is closed with `slybot_fewitems_scraped` outcome.

By default, the period of time is 3600 seconds and the threshold is 200 items scraped. This means if less than 200 items were scraped in the last 3600 seconds, the job will close.

You can set the period in seconds with the `SLYCLOSE_SPIDER_CHECK_PERIOD` setting, and the threshold number of items with the `SLYCLOSE_SPIDER_PERIOD_ITEMS` setting.

6.1 What are samples?


When the crawler visits a page, it matches the page against each sample. Samples with more annotations take precedence over those with less. If the page matches a sample, it will use the sample's annotations to extract data. Assuming all required fields are filled, it will yield an item. Spiders consist of one or more samples and each sample is made up of annotations that define the elements you wish to extract. Within the sample you define the item you want to extract and mark required fields for that item.





6.2 What are annotations?


An annotation defines the location of a piece of data on the web page and how it should be used by the spider. Typically an annotation maps some element on the page to a particular field of an item, but there is also the option to mark the data as being required without storing the data in an item. It's possible to map attributes of a particular element instead of the content if this is required, for example you can map the `href` attribute of an anchor link rather than the text.

6.3 Annotations

6.3.1 Creating annotations

You can create annotations by clicking an element on the page with the appropriate tool selected. You should use the wand () most of the time as it will select the appropriate tool automatically. The following tools are available:

-  - Select the most appropriate tool when clicking on an element
-  - Select an element
-  - Add an element
-  - Remove an element

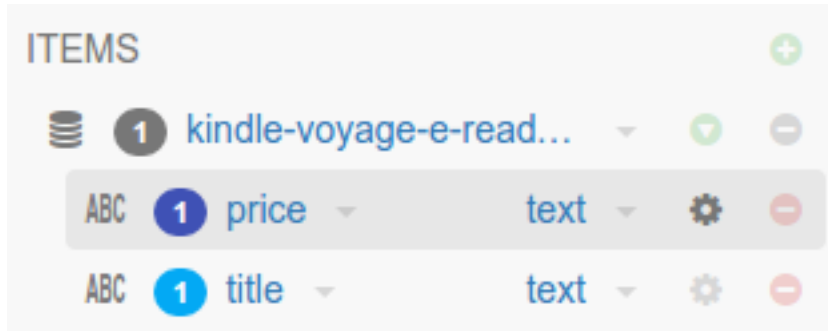
-  - Add repeating element

6.3.2 Extractors

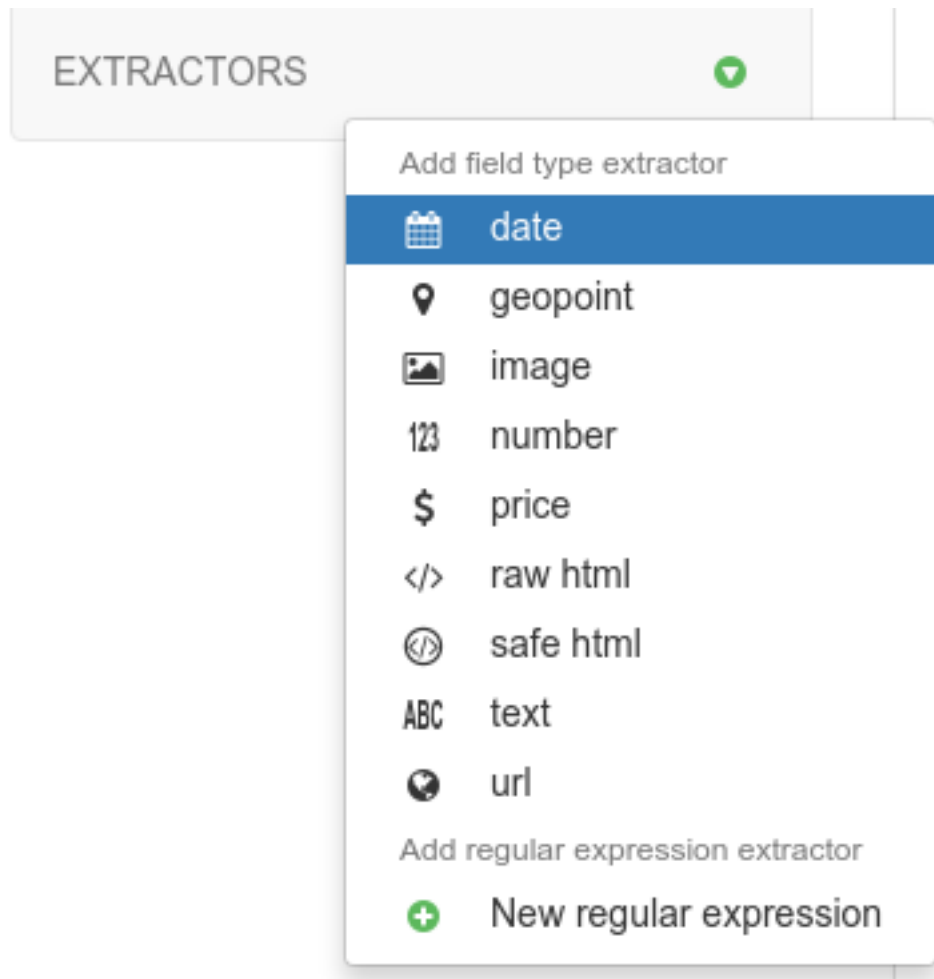
You can also add extractors to annotations. Extractors let you use regular expressions or a pre-defined type to further refine data extracted from a page.

For example, assume there's an element that contains a phone number, but it has additional text that you don't need. In this scenario you could add an extractor to retrieve only the phone number instead of the full text.

You can define the extractor for a particular field by clicking in the gear icon right after the field type:



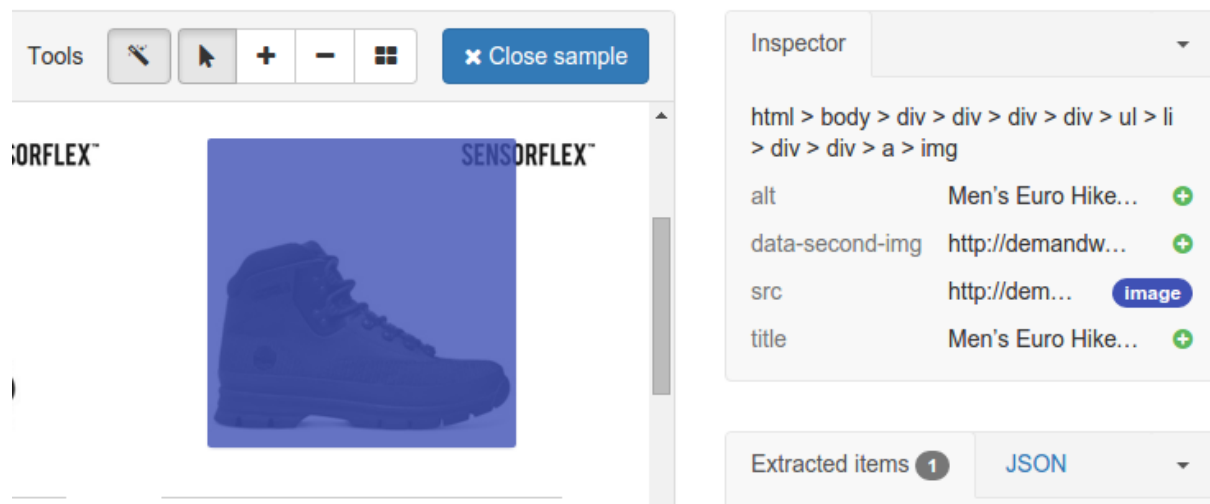
And then you can select use any built-in extractors or create your own extractor via regular expressions:



6.3.3 Multiple fields

It's possible to extract multiple fields using a single annotation if there are several properties you want to extract from an element. For example, if there was an image you wanted, you could map the `src` attribute that contains the image URL to one field, and the `alt` attribute to another.

You can do it in the `Inspector` panel in the top left of the screen:



Just click the + button right after an attribute to add a new field based on the same annotation.

6.4 Multiple samples

It's often necessary to use multiple samples within one spider, even if you're only extracting one item type. Some pages containing the same item type may have a different layout or fields missing, and you will need to accommodate for those pages by creating a sample for each variation in layout.

6.4.1 Sample precedence

The more annotations a sample has, the more specific the data being extracted and therefore less chance of a false positive. For this reason, samples with more annotations take precedence over those with less annotations. If a subset of samples contains equal number of annotations per sample, then within that subset samples will be tried in the order they were created from first to last. In other words, samples are tried sequentially in order of number of annotations first, and age second.

If you are working with a large number of samples, it may be difficult to ensure the correct sample is applied to the right page. It's best to keep samples as strict as possible to avoid any false matches. It's useful to take advantage of the `Required` option from item fields and annotate elements that will always appear on matching pages to reduce the number of false positives.

Check this example to learn how to do it: [Using Multiple Samples to Deal with Different Layouts](#).

An item refers to a single item of data scraped from the target website. A common example of an item would be a product for sale on an e-commerce website. It's important to differentiate **item** and **item definition**. In Portia, an item definition or item type refers to the schema of an item rather than the item itself. For example, `book` would be an item definition, and a specific book scraped from the website would be an item. An item definition consists of multiple fields, so using the example of a product you might have fields named `name`, `price`, `manufacturer` and so on. We use annotations to extract data from the page into each of these fields.

To ensure certain fields are extracted, you can set the **Required** flag on each required field. Portia will discard an item if any required fields are missing. Portia will also remove any duplicate items by default.

In some cases you may have fields where the value can vary despite being the same item, in which case you can mark them as **Vary**. This will ignore the field when checking for duplicates. It's important to only use **Vary** when necessary, as misuse could easily lead to duplicate items being stored. The `url` field is a good example of where **Vary** is useful, as the same item may have multiple URLs. If the `url` field wasn't marked as **Vary**, each duplicate item would be seen as unique because its URL would be different.

7.1 Field types

You can set a field's type to ensure it will only match that kind of data. The following field types are available:

type	description
text	Plain text. Any markup is stripped and text within nested elements is also extracted.
number	A numeric value e.g. 7, 9.59.
image	An image URL. In most cases you will want to map an <code>img</code> element's <code>src</code> attribute.
price	The same as <code>number</code> , a numeric value.
raw html	Non-sanitized HTML.
safe html	Sanitized HTML. See below for more details.
geopoint	The same as <code>text</code> .
url	A URL.
date	A date value parsed by <code>dateparser</code> . Won't work if the annotated element has non-date text.

The `safe_html` field type keeps the following elements: `br`, `p`, `big`, `em`, `small`, `strong`, `sub`, `sup`, `ins`, `del`, `code`, `kbd`, `samp`, `tt`, `var`, `pre`, `listing`, `plaintext`, `abbr`, `acronym`, `address`, `bdo`, `blockquote`, `q`, `cite`, `dfn`, `table`, `tr`, `th`, `td`, `tbody`, `ul`, `ol`, `li`, `dl`, `dd`, `dt`.

All other elements are discarded, with the exception of header tags (`h1`, `h2` ... `h6`) and `b` which are replaced with `strong`, and `i` which is replaced with `em`. Whitelisted elements contained within non-whitelisted elements will still be retained, with the exception of elements contained within a `script`, `img` or `input` element. For example, `<div><code>example</code></div>` would extract to `<code>example</code>`, whereas `<script><code>example</code></script>` would be discarded completely.

8.1 How do I use Crawlera with Portia?

Portia spiders are standard Scrapy spiders, so you can enable the `middleware` in your project's *settings.py*.

8.2 Does Portia support AJAX based websites?

Yes.

8.3 Does Portia work with large JavaScript frameworks like Ember?

Backbone, Angular, and Ember have all been thoroughly tested using Portia, and in most cases should work fine. React based websites aren't supported yet but we're working on it.

8.4 Does Portia support sites that require you to log in?

Yes, you can set credentials in your spider's crawling configuration.

8.5 Does Portia support content behind search forms?

No, but we plan on adding support in the near future.

CHAPTER 9

Indices and tables

- `genindex`
- `modindex`
- `search`