

# **Practice of Efficient Data Collection via Crowdsourcing: Aggregation, Incremental Relabelling, and Pricing**

Setting up and running label collection projects  
**Instruction**

Yandex Toloka for requesters:

<https://toloka.ai/>

For dataset:

[http://tlk.s3.yandex.net/relsubstitutes/dataset\\_Y.tsv](http://tlk.s3.yandex.net/relsubstitutes/dataset_Y.tsv)

where Y is any number from 1 to 10

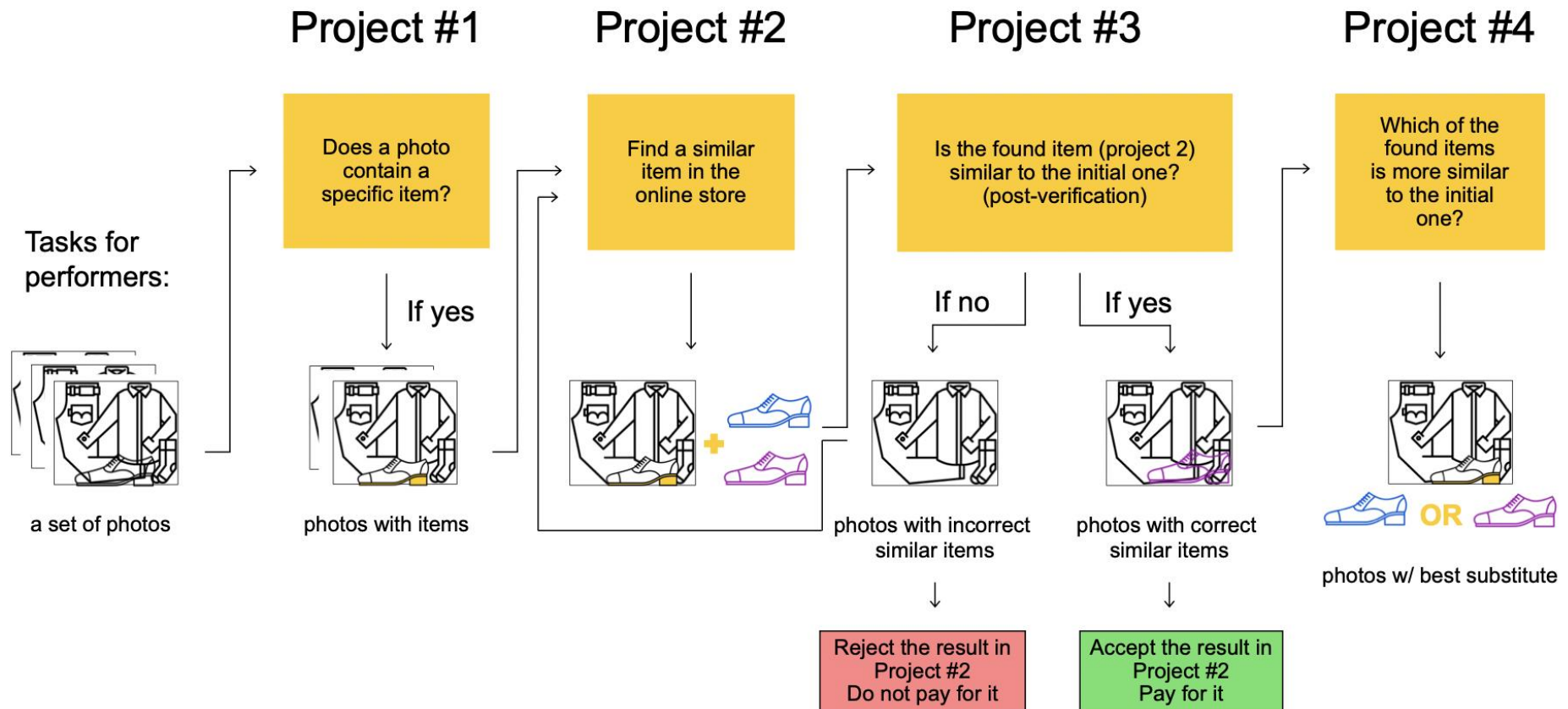
Tutorial slides:

<https://research.yandex.com/tutorials/crowd/wsdm-2020>

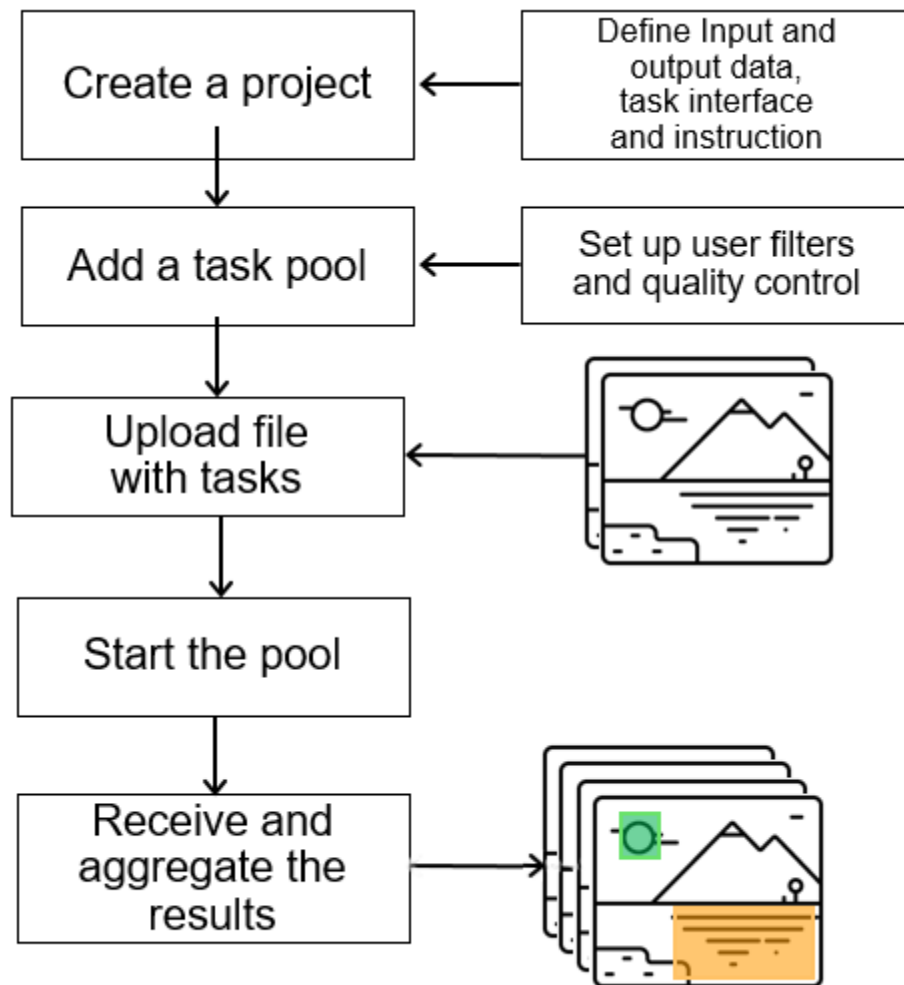
# Table of contents

Suggested pipeline .....	3
Project creation. Main steps .....	4
Key types of instances in Yandex.Toloka .....	4
<b>Project #1</b> Does the photo contain an item? .....	5
Project creation.....	6
Pool creation .....	9
Preparing and uploading a file with tasks .....	11
Receiving responses .....	14
<b>Project #2</b> Find a similar item in an online store .....	15
Project creation.....	16
Pool creation .....	19
Preparing and uploading a file with tasks .....	23
<b>Project #3</b> Does the item found look similar to the initial one? .....	24
Project creation (similar to the 1 <sup>st</sup> project).....	25
Pool creation .....	28
Preparing and uploading a file with tasks .....	30
Receiving responses .....	34
Upload reviewed results .....	35
Review assignments in the interface (another way of results validation) .....	36
<b>Project #4</b> Which item is more similar?.....	37
Project creation.....	38
Pool creation .....	42
Preparing and uploading a file with tasks .....	44
Receiving responses .....	45
<b>Appendix:</b> Expanded code of the projects .....	46

# Suggested pipeline



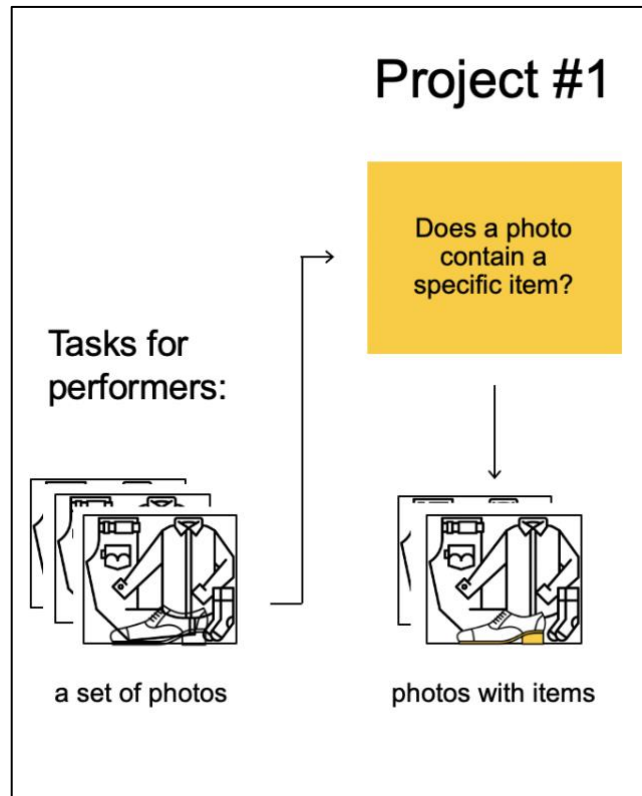
## Project creation. Main steps



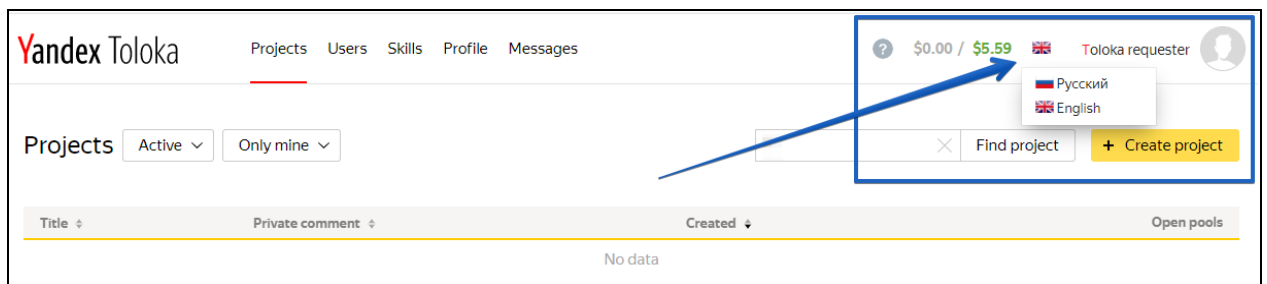
## Key types of instances in Yandex.Toloka

Project	Pool	Task
<ul style="list-style-type: none"> <li>› Defines the structure of tasks</li> <li>› Defines how to perform them</li> </ul>	<ul style="list-style-type: none"> <li>› Is a batch of tasks</li> <li>› Defines access of performers</li> </ul>	<ul style="list-style-type: none"> <li>› A particular input data</li> <li>› Results for it from performers</li> </ul>
<b>Configure in a project:</b> <ul style="list-style-type: none"> <li>› Input and output data types</li> <li>› Task interface</li> <li>› Task instruction</li> </ul>	<b>Configure in a pool:</b> <ul style="list-style-type: none"> <li>› Performer filters</li> <li>› Quality control mechanisms</li> <li>› Overlap settings</li> </ul>	

# Project #1 Does the photo contain an item?

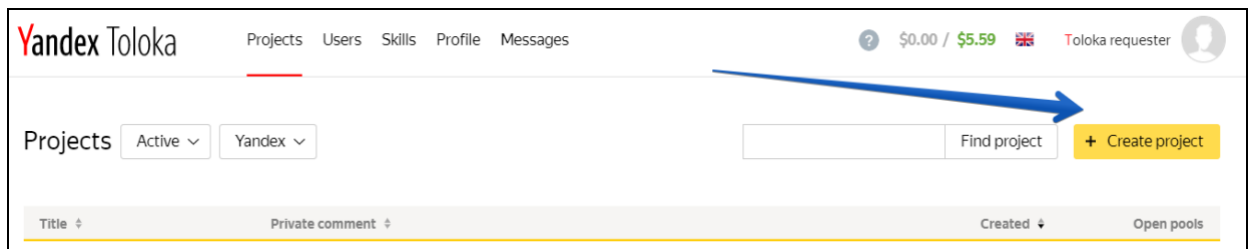


**Important:** Before you start using Toloka, make sure that the **English** language is selected.



## Project creation

1. Click the button **+ Create project**




2. Choose the **Image classification** template.



3. Enter a clear project name and description.

**Important:** It will be visible for real people.

The screenshot shows the project creation form. It has three main sections: 'PROJECT NAME' with the text 'Are there shoes in the picture?', 'DESCRIPTION' with the text 'Look at the picture and tell whether there are shoes in it.', and 'INSTRUCTIONS' with a rich text editor containing the text: 'Look at the photo and decide, whether there are shoes in the photo. If yes, click "YES" If no, click "NO" For example, there are shoes in the photo. Therefore, the correct answer is "YES"'. There is a small image of a person's head at the bottom right of the instructions area.


4. Write short and simple instructions. **To include an image in the instruction just paste the link from the dataset provided by pressing  button.**

**Instructions**

Look at the photo and decide, whether there are **shoes** in the photo.

If yes, click "YES"  
If no, click "NO"

For example, there are shoes in the photo. Therefore, the correct answer is "YES"



5. Define parameters for the [input and output data](#):

- The **"image"** input data field with the link type will be used to pass the image links to the performers.  
*You will be able to upload the file with links to the pool later.*
- The **"result"** field will be used to receive performer's responses.
- The **"like"** field in the template is used to pass the response to the question *"Do you like the photo?"*. **Our project doesn't require this checkbox, so you don't need an output field for it. Let's remove it.**

Input data	Output data
<div>image (URL)</div>	<div>result (string)</div>
	<div>like (boolean) <b>DELETE</b></div>
<div>Add field</div>	<div>Add field</div>

6. Create the task interface in the HTML block.

- Delete the line with the checkbox component:  

```
{{field type="checkbox" name="like" label="Do you like the photo?" hotkey="q"}}
```
- Add a question: does the image include a certain object? Example:  

```
<div>Are there <b>shoes</b> in the picture?</div>
```
- Replace **"label"** with your response options (change "Good" to "Yes" and "Bad" to "No"). Example:

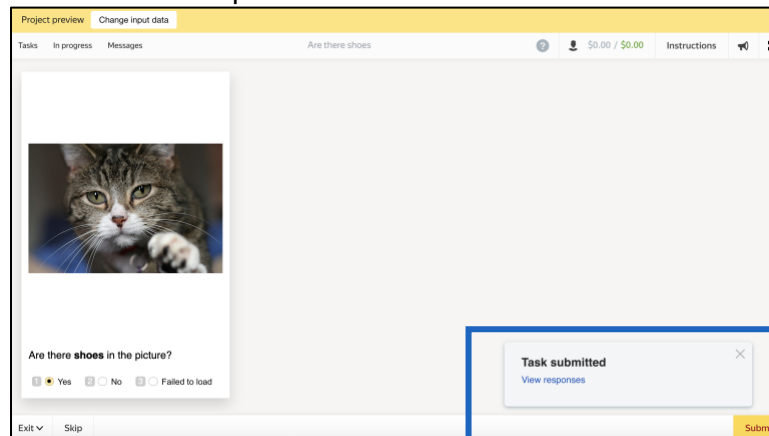
```
{{img src=image width="100%" height="400px"}}
<p> Are there <b>shoes</b> in the picture?</p>
{{field type="radio" name="result" value="Yes" label="Yes"}}
{{field type="radio" name="result" value="No" label="No" }}
{{field type="radio" name="result" value="404" label="Picture not found" }}
```

7. Leave JavaScript and CSS block unchanged.

8. Click the **Preview** button to see the performer's view of the task.

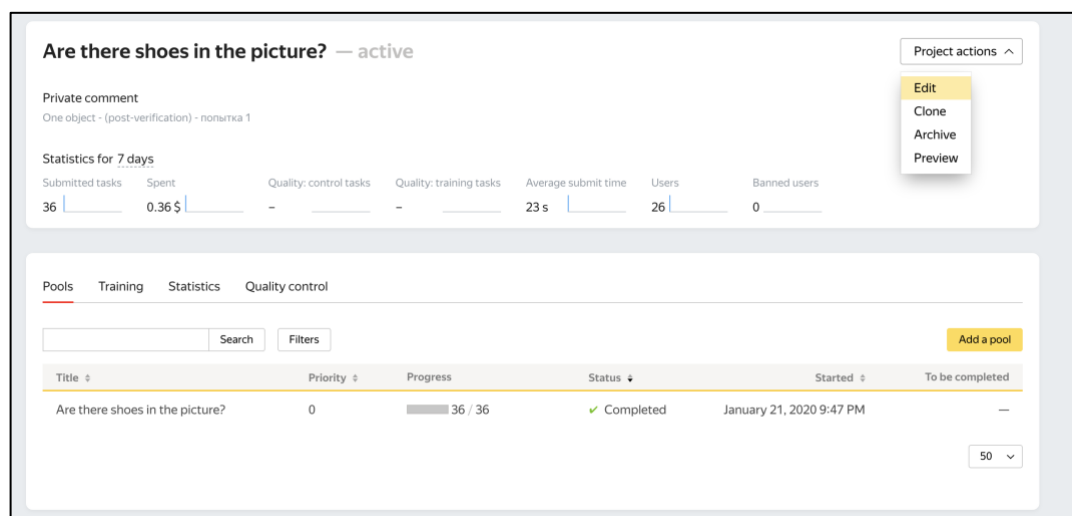
*You will see tasks with standard pictures on the page. You will set the number of tasks per page when configuring a pool.*

9. Select the radio buttons in the preview and make sure that the task can be completed.



10. Click **Save** button to save the project.

**Note.** To edit project parameters, click the button in the list of projects or **Project actions** → **Edit** on the project page.





## Pool creation

1. Click **Add pool**.
2. Give the pool any convenient name and description. You are the only one who can see them.
3. Specify the [pool parameters](#):
  - Set the price per task page (for example, \$0.01).

**Price per task suite**

You can add one or more tasks to the page. Enter the total price for all tasks on the page

PRICE IN US DOLLARS ? 0.01

FEE ? 0.005

+ Dynamic pricing

4. Set up user [filters](#).
  - Select English-speaking performers using the **Language = English** filter.

**Performers**

Filter performers who can access the task. Toloka has users from different countries, so don't forget to filter by language and region. [Learn more](#)

ADULT CONTENT ? No

Add filter Create skill

PROFILE

Languages = English

5. Set up [quality control](#): [Control tasks](#). Ban performers who give incorrect responses to control tasks. Example:

**Quality control**

Add rules to get more accurate responses. All rules work independently.

NON-AUTOMATIC ACCEPTANCE ? No

REVIEW PERIOD IN DAYS

CAPTCHA FREQUENCY ? None

CONTROL TASKS ?

Recent values to use Items

If number of responses ≥ 3

and correct responses (%) < 60

then ban on project

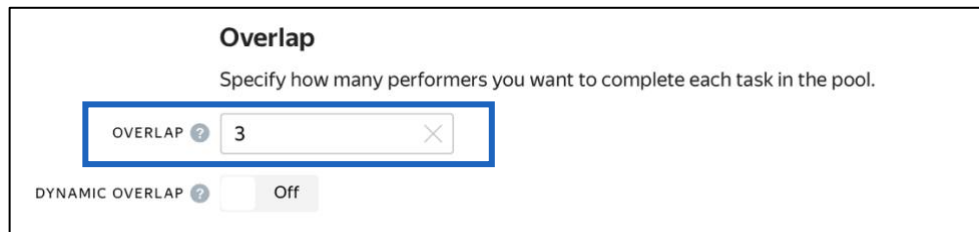
10 days

Control tasks

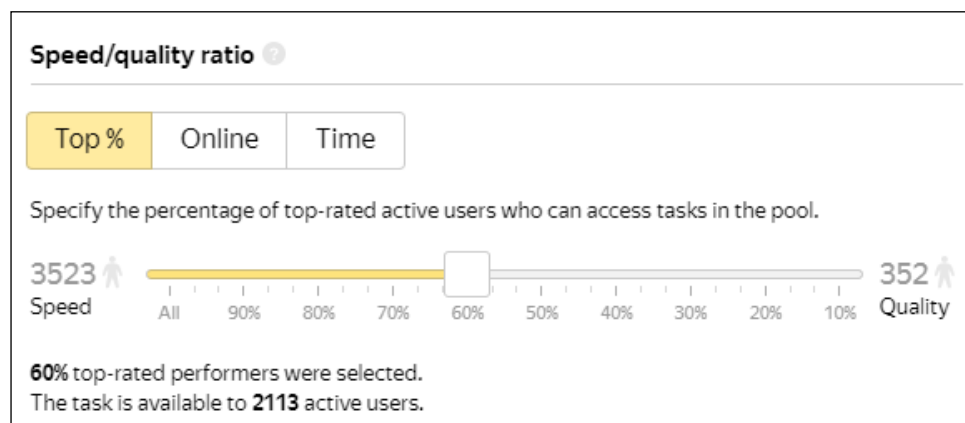
This rule will be triggered when the performer completes 3 control tasks in the pool. If the performer gives at least 3 responses to the control tasks and the percentage of correct responses is less than 60%, they lose access to the project for 10 days. If the percentage of correct responses is over 60%, the performer can pass to the next task page. The rule will be triggered after the next control task.

Optionally, add [other quality control rules](#).

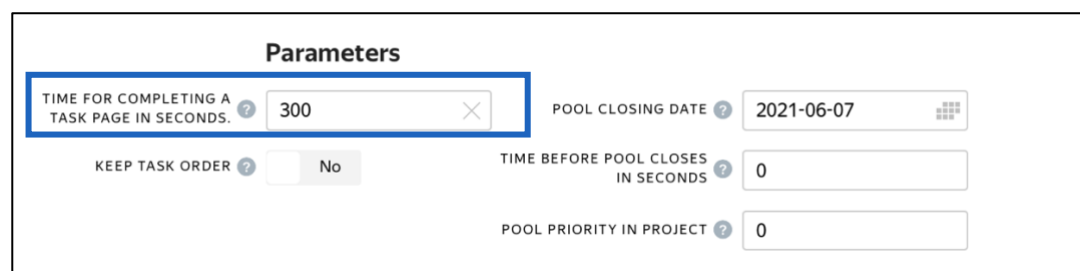
6. Overlap. This is the number of users who will complete the same task. For example, 3.



7. Optionally, specify the percentage of top-rated performers in the [Speed / Quality ratio](#).  
**Important:** This can slow down pool completion.



8. Time allowed for completing a task page (for example, 300 seconds).

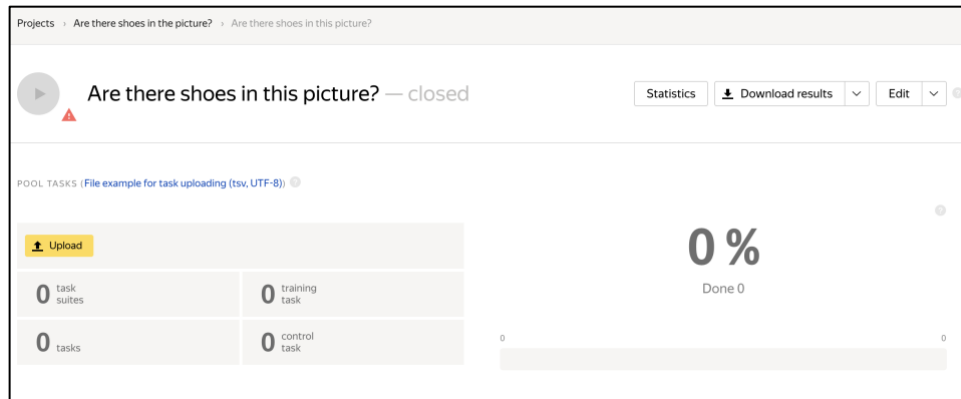


9. Save the pool.

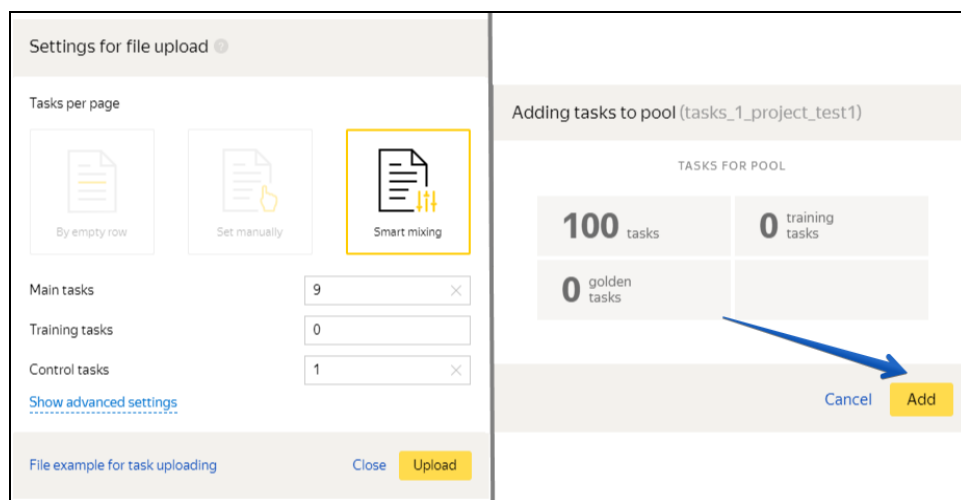
## Preparing and uploading a file with tasks

1. Download TSV-file with images by link that you were provided at the beginning of the practice session.
2. [Upload pool tasks](#) from this file.

**Important:** If you changed the name of the input field, change it in the file as well



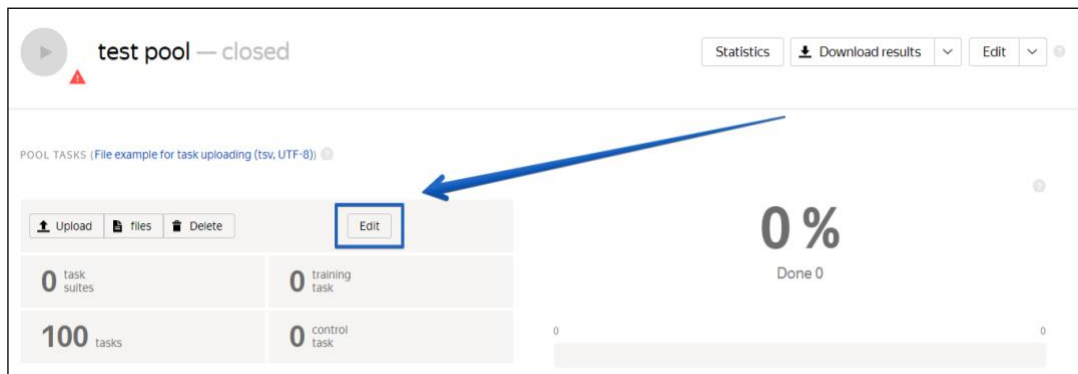
Select [Smart mixing](#) and specify the number of tasks per page.  
For example: 9 main tasks and 1 control task.



3. [Create control tasks](#).

**Note.** Control tasks are tasks with the correct response known in advance. They are used to track the performer's quality of responses. The performer's response is compared to the response you provided. If they match, it means the performer answered correctly.

- Click **Edit** → **Create control tasks**.



### Edit tasks

Use main tasks as a starting point to create control tasks or training tasks.  
Control tasks are for checking the quality of responses from performers. They contain correct responses to compare with actual responses.  
Training tasks are for teaching performers how to complete tasks. They contain correct responses and hints.  
[Learn more](#)

Main 100   Control tasks 0   Training tasks 0

**Create control tasks**   Create training tasks   Download

ID	Overlap	Responses from performers	Last updated
...1975eb01	3	0	07/26/2019 22:18:06
...1975eabe	3	0	07/26/2019 22:18:06
...1975ead1	3	0	07/26/2019 22:18:06

- Check the "result" output field that is used to match the user response to the control one, select the response and click **Save and go to next**.

Projects > Are there shoes > Are there trousers > Uploaded tasks > Edit tasks

Control tasks: 3

**Create control task**

1. Specify the correct answers  
2. Select the fields to include

Field	Value
result	Yes

Are there shoes in the picture?

☒ Yes ☐ No ☐ Failed to load

**Save and go to next**

Distribution of correct responses for control tasks

result

100% Yes

Enter correct responses for your control tasks. In small pools, control tasks should account for 10-20% of all tasks.

**Tip.** Make sure to include different variations of correct responses in equal amounts. Open the **Control tasks** → **Distribution of correct responses for control tasks** tab.

Projects > Does the image contains t... > test pool > Uploaded tasks

### Edit tasks

Use main tasks as a starting point to create control tasks or training tasks.  
Control tasks are for checking the quality of responses from performers. They contain correct responses to compare with actual responses.  
Training tasks are for teaching performers how to complete tasks. They contain correct responses and hints.  
[Learn more](#)

Main 90 **Control tasks 10** Training tasks 0

Create control tasks Create training tasks Download

ID	Overlap	Responses from performers	Last updated
...19f9102c	3	0	07/26/2019 21:45:27
...19f9101b	3	0	07/26/2019 21:45:27

Main 90 **Control tasks 10** Training tasks 0

Create from main tasks Download

ID	Overlap	Responses from performers	Correct responses, %	Last updated
...19f90ff6	∞	0		07/26/2019 21:45:27
...19f90ff0	∞	0		07/26/2019 21:45:27
...19f90ff1	∞	0		07/26/2019 21:45:27

Distribution of correct responses for control tasks

result

40% BAD

60% OK

- Save the markup and check the number of control tasks.

test pool — closed

Statistics Download results Edit

POOL TASKS (File example for task uploading (tsv, UTF-8))

Upload files Delete Edit Preview

~30 task suites 0 training task

90 tasks **10 control task**

0 % Done 0

4. Start the pool.

**Important.** Remember that real Toloka performers will complete the tasks. Double check that everything is correct with configuration of your project before you start the pool.

test pool — closed

Statistics Download results Edit

POOL TASKS (File example for task uploading (tsv, UTF-8))

Upload files Delete Edit Preview

~30 task suites 0 training task

90 tasks 10 control task

0 % Done 0

## Receiving responses

**Disclaimer: Aggregation takes from 5 to 20 minutes. During this time, you can start configuring your next project. Refresh the Operations page to check progress.**

1. Wait until the pool is completed. Refresh the pool page to check progress.
2. Click the arrow next to the **Download results** button and run aggregation using the **Dawid-Skene model**.

The screenshot shows the 'pool' page with a status of 'closed'. A dropdown menu is open next to the 'Download results' button, with 'Dawid-Skene aggregation model' selected. A progress bar at the bottom indicates 100% completion. The page also displays task statistics: 30 task suites, 0 training tasks, 90 tasks, and 10 control tasks.

3. Go to the operations list and wait until aggregation finishes.  
**Note.** During this time, you can start working on your next project. Refresh the Operations page to check progress.

The screenshot shows a green notification bar at the top of the page with the text 'Assignment aggregation started successfully. View operations list.' A blue arrow points from the 'View operations list' link to the 'Operations' page screenshot below.

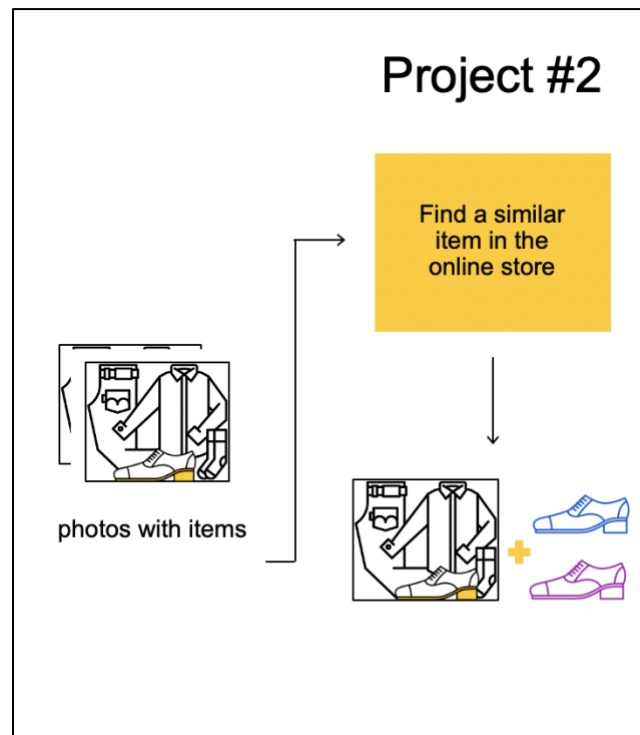
4. When aggregation is complete, download the TSV file with the results.

The screenshot shows the 'Operations' page with a table of tasks. The task 'Dawid-Skene aggregation model' is shown with a status of 'Success' and a progress of 100%. A blue arrow points from the 'Download' link in the 'Files' column to the 'Download results' button in the pool page screenshot above.

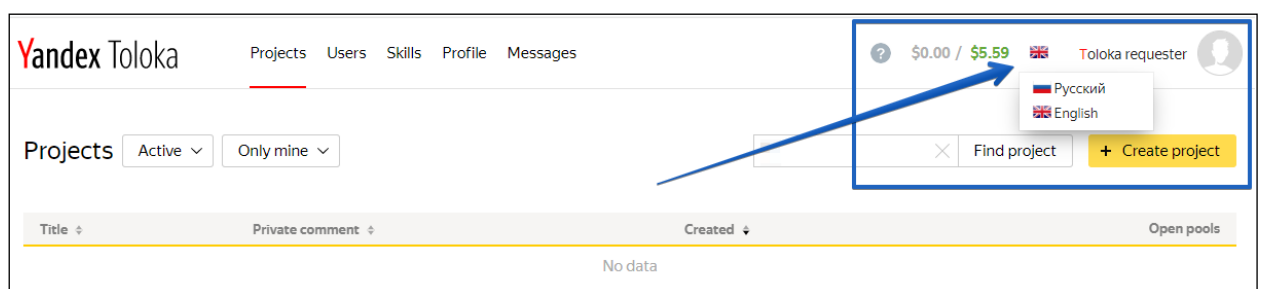
Id	Type	Started	Finished	Progress	Status	Files
0f1391...	Dawid-Skene aggregation model	06/26/2019 13:52:16	06/26/2019 13:57:12	100%	Success	<a href="#">Download</a>

5. Use this file to [prepare data for Project #2](#).

# Project #2 Find a similar item in an online store



**Important:** If you just started using Toloka, make sure that the English language is selected.







The code for specifications are:

#### Input data:

```
{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}
```

#### Output data:

```
{
  "button": {
    "type": "boolean",
    "hidden": false,
    "required": true,
    "allowed_values": [
      true
    ]
  },
  "found_link": {
    "type": "string",
    "hidden": false,
    "pattern": "https://www.marksandspencer.com/.*",
    "required": true
  },
  "found_image": {
    "type": "file",
    "hidden": false,
    "required": true
  }
}
```

#### 5. Create the task interface.

- **Delete the whole HTML code** in the template, and instead add the following code to show the initial item to the performers:

```
{{img src=image width="50%" height="400px"}}
<div class='answers'>
  <p>Find the same <b>shoes</b> on Marks and Spencer</p>
  {{field type="button-clicked" name="button" label="Marks and Spencer"
href="https://www.marksandspencer.com" action=true}}

  <p>Shoes must be the same color and the same style.</p>
  <p>Paste the link here</p>
  {{field width="100%" type="input" name="found_link"}}
  <p>Upload the image here</p>
  <div>
    {{field width="100%" type="file-img" name="found_image" preview=true}}
  </div>
</div>
```


- #### 6. Now we need to check whether performer is going to submit a valid link and an image. To check it, **DO NOT delete any of JS code**

In case you are having trouble previewing call our team or check the expanded code in Appendix.

7. Add the following code in CSS field to set the images sizes proportionally:

```
.task {  
  display: block;  
  height: 500px;  
  width: 800px;  
}  
.img {  
  float: left;  
  width: 50%;  
}  
.answers {  
  float: left;  
  width: 40%;  
  margin: 5%;  
}
```

8. Click the **Preview** button to see the performer's view of the task. **!** In this particular case you will not be able to submit the assignment in **Preview** as the image you will be trying to upload cannot be uploaded while the pool is still closed.




Find the same **shoes** on Marks and Spencer

Marks and Spencer

Shoes must be the same color and the same style.

Paste the link here

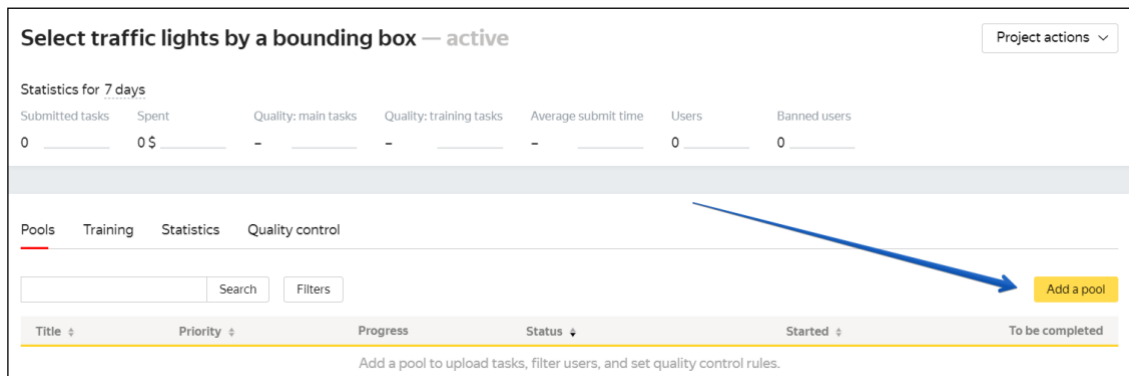
Upload the image here



9. Click **Save** button to save the project.

## Pool creation

1. Click **Add pool**.



Select traffic lights by a bounding box — active

Project actions ▾

Statistics for 7 days

Submitted tasks	Spent	Quality: main tasks	Quality: training tasks	Average submit time	Users	Banned users
0	0 \$	-	-	-	0	0

Pools Training Statistics Quality control

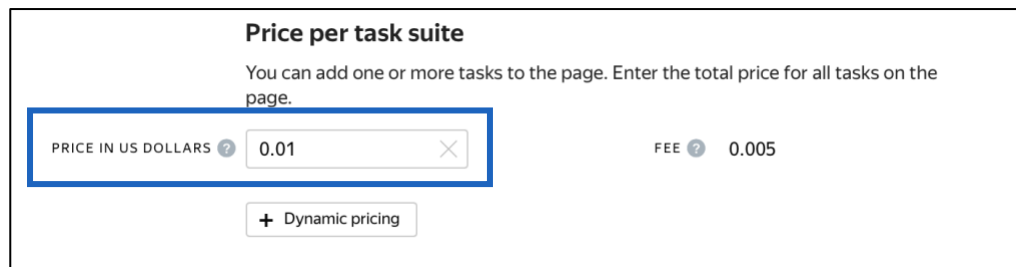
Search Filters

Add a pool

Title	Priority	Progress	Status	Started	To be completed
-------	----------	----------	--------	---------	-----------------

Add a pool to upload tasks, filter users, and set quality control rules.

2. Give the pool any convenient name and description. You are the only one who can see them.
3. Specify the [pool parameters](#):
  - Price per task page (for example, \$0.01)



Price per task suite

You can add one or more tasks to the page. Enter the total price for all tasks on the page.

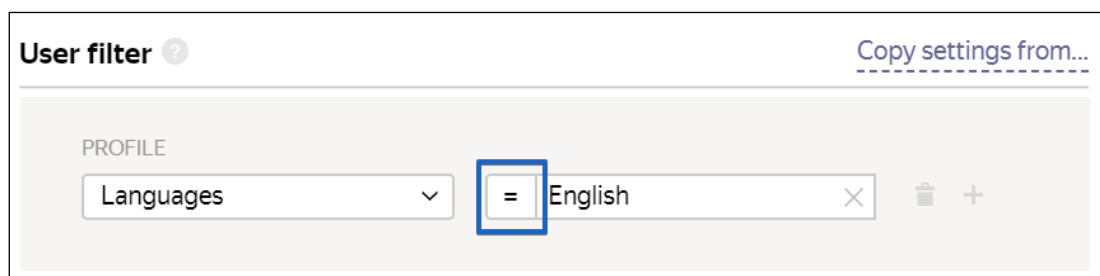
PRICE IN US DOLLARS 0.01

FEE 0.005

+ Dynamic pricing

4. Set up user [filters](#).

- Select English-speaking performers using the **Language = English** filter.



User filter ?

Copy settings from...

PROFILE

Languages ▾ = English

- Create the "**Found\_shoes**" [skill](#) that will be assigned to users after they complete the pool tasks. You will use this skill to prevent these users from checking tasks in the next project. Click **Create skill**:



Users filter ?

Copy settings from...

Add filter ▾ Create skill

- Enter the skill name and add a description if desired. You are the only one who will see it. Leave the skill **private**, as it is by default and click **Add**.

**Add skill**

TITLE  
Found\_shoes

DESCRIPTION

Public? ☐ No

Cancel Add

5. Turn on the **Non-automatic acceptance** option and enter the number of days for checking in the **Deadline** field (for example, 7).

**Quality control**  
Add rules to get more accurate responses.  
All rules work independently.

NON-AUTOMATIC ACCEPTANCE ☒ Yes

REVIEW PERIOD IN DAYS 7

CAPTCHA FREQUENCY None

6. Set up [quality control](#):

- Resend the rejected tasks for completion. Add the [Recompletion of rejected assignments](#):

**Quality control**

RECOMPLETION OF REJECTED ASSIGNMENTS

If assignment becomes rejected

then extend overlap by 1

☒ Open pool if closed

- [Submitted responses](#). Add a rule to mark users who completed at least one task in the pool.

SUBMITTED RESPONSES ?

If submitted assignments ≥ 1

then assign skill value found\_shoes 1

**Tip.** If the skill you created doesn't appear in the drop-down list, save the pool, and then open it for editing again.

- Add [Fast responses](#) rule to block those who provide information suspiciously fast

FAST RESPONSES ?

Recent values to use items

Minimum time per task suite 60

If number of fast responses ≥ 1

then ban on project

10 days

fast responses

- Add [Results of assignments review](#) rule to ban those who brought results of improper quality

RESULTS OF ASSIGNMENT REVIEW ?

Recent values to use items

If rejected responses (%) ≥ 1

then ban on requester

20 days

rejected assignment

Optionally, add [other quality control rules](#).

**Tip.** Control tasks and majority vote are not used in this type of project, because performer's links and photos that she will provide must exactly match the reference, which is practically impossible.

7. Overlap. This is the number of users who will complete the same task. Because we want various options for each photo, put overlap equal to 3.

### Overlap

Specify how many performers you want to complete each task in the pool.

OVERLAP ?

3

×

DYNAMIC OVERLAP ?

☐ Off

8. Optionally, specify the percentage of top-rated performers in the [Speed / Quality ratio](#).

### Speed/quality ratio ?

Top %

TOP N

Time

Specify the percentage of top-rated active users who can access tasks in the pool.

3523

Speed

All

90%

80%

70%

60%

50%

40%

30%

20%

10%

Quality

352

**60%** top-rated performers were selected.  
The task is available to **2113** active users.

**Important:** This can slow down pool completion.

9. Time allowed for completing a task page (for example, 300 seconds)

### Parameters

TIME FOR COMPLETING A TASK PAGE IN SECONDS. ?

300

×

POOL CLOSING DATE ?

2021-06-04

⌵

KEEP TASK ORDER ?

☐ No

TIME BEFORE POOL CLOSES IN SECONDS ?

0

POOL PRIORITY IN PROJECT ?

0

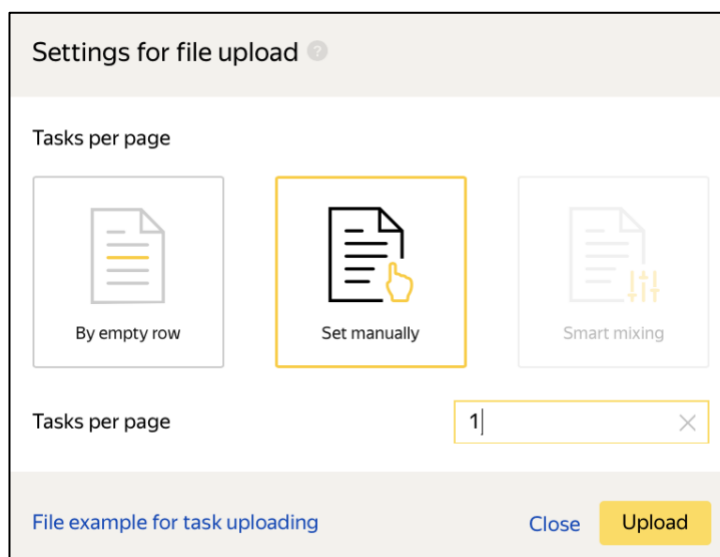
10. Save the pool.

## Preparing and uploading a file with tasks

1. Open the file with [aggregated results from the project #1](#).
2. Select only images suitable for highlighting (**OK** answers or another value if you have changed it in the "**result**" field). Use a text editor or a spreadsheet editor.
3. Copy the column with the selected links to a new page or document and give a name to the **INPUT:image** column.

**Important:** If you changed the input field name in the project to something other than "image", change the name in the file as well: INPUT:<your field name>.

4. Save the file in TSV format.
5. [Upload the file](#) to the pool by selecting **Set manually**. Set 1 task per page.



Settings for file upload ?

Tasks per page

By empty row

Set manually

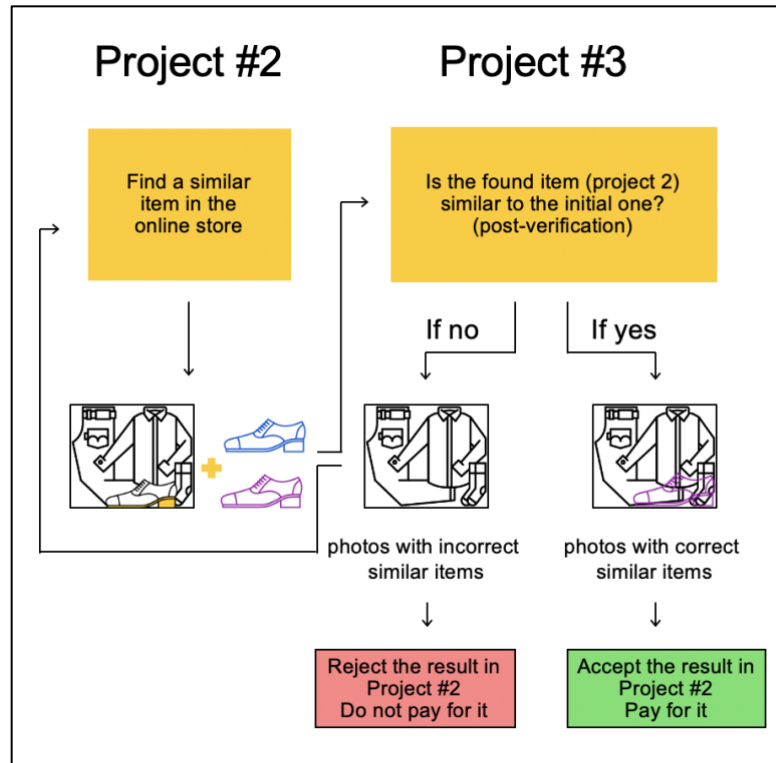
Smart mixing

Tasks per page 1

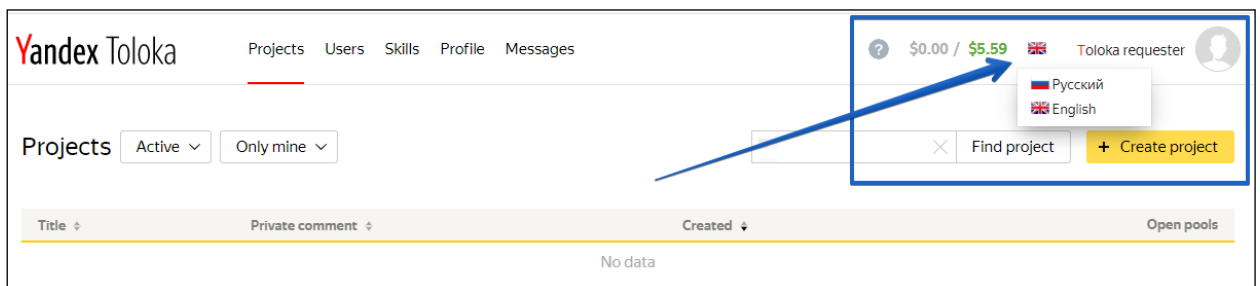
[File example for task uploading](#) [Close](#) [Upload](#)

6. Start the pool.

# Project #3 Does the item found look similar to the initial one?



**Important:** If you just started using Toloka, make sure that the **English** language is selected.



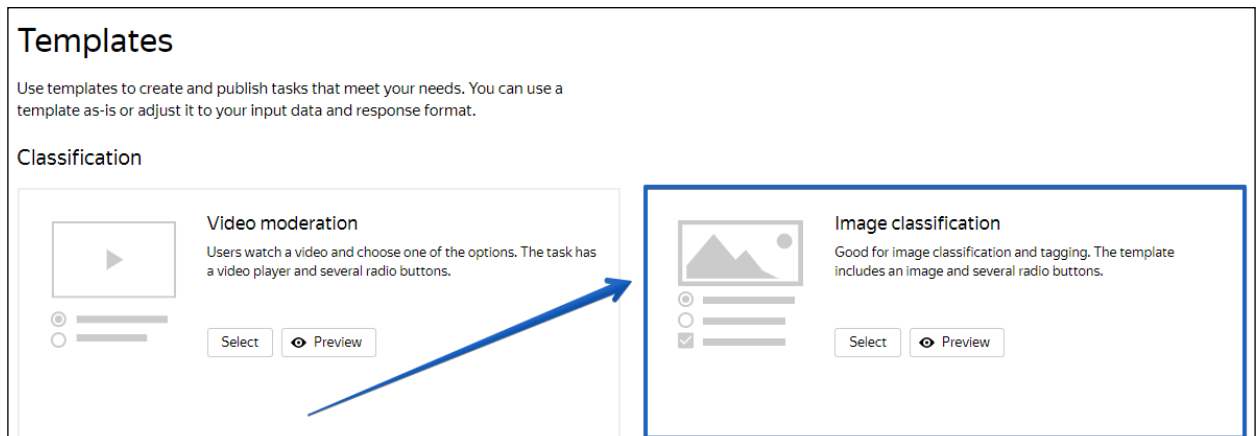


## Project creation (similar to the 1<sup>st</sup> project)

1. Click the button **+ Create project**



2. Choose the **Image classification** template.



3. Enter a clear project name and description. **Important:** It will be visible for real people.

A screenshot of the project creation form. It has three main sections: 'PROJECT NAME', 'DESCRIPTION', and 'INSTRUCTIONS'. The 'PROJECT NAME' field contains the text 'Do the shoes look similar?'. The 'DESCRIPTION' field contains the text 'Take a look at two pairs of shoes and decide whether they look similar or not.'. The 'INSTRUCTIONS' section has a rich text editor with a toolbar. The text entered in the instructions field is: 'Take a look at the pictures, which will show two pairs of shoes. **Decide whether they look similar or not.** The shoes will look similar if they are the same or similar color, fabric, length and style. If you do not see a pair of shoes in any of the pictures please click "NO" button.'

4. Write short and simple instructions.
5. Define parameters for the **input and output data**:

Input:

- The **"image"** input data field with the "url" type will be used to pass the initial image links to the performers.
- The **"found\_link"** field with the "url" type will allow performers can go to the website.
- The **"assignment\_id"** field with the "string" type, will be used to pass the number of the completed task.

Output:

- Leave “**result**” as it is.

The code for specifications is:

Input data:

```
{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "found_link": {
    "type": "url",
    "hidden": false,
    "required": true
  },

  "assignment_id": {
    "type": "string",
    "hidden": true,
    "required": true
  }
}
```

Output data:

```
{
  "result": {
    "type": "string",
    "hidden": false,
    "required": true
  }
}
```

6. Delete the whole **HTML code** in the template, and add the following one.

```
{{img src=image height="400px"}}
{{iframe src=found_link height="600px"}}

<p>Check that the uploaded image matches the product in the store.</p>
  {{button label="Check the item" href=found_link action=true}}

<p>Are these <b>shoes</b> similar to each other?</p>
  <p>Shoes must be the same color and the same style.</p>
  {{field type="radio" name="result" value="Yes" label="Yes"}}
  {{field type="radio" name="result" value="No" label="No"}}

</div>
```

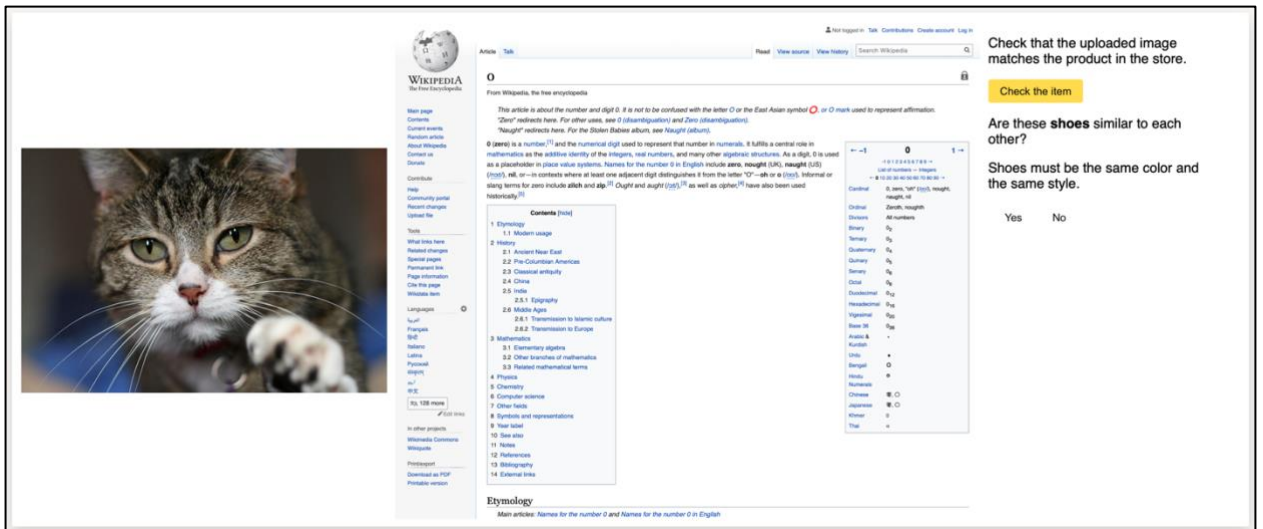
7. Leave the JS block unchanged.
8. In the CSS block paste the following (don't forget the other half on the next page):

```
.task {
display: block;
min-height: 620px;
width: 100%;
box-sizing: border-box;
width: calc(100% - 100px);
}

.img {
float: left;
width: 30%;
}
```

```
.text {
float: left;
width: 18%;
margin-left: 10px;
}
```

9. Click the **Preview** button to see the performer's view of the task.



10. Select the radio buttons in the preview and make sure that the task can be completed.

11. Click **Save** button to save the project.

**Note.** To edit project parameters, click the button in the list of projects or **Project actions** → **Edit** on the project page.

## Pool creation

1. Click **Add pool**.
2. Give the pool any convenient name and description. You are the only one who can see them.
3. Specify the [pool parameters](#):
  - Set the price per task page (for example, \$0.01).

The screenshot shows a configuration box titled "Price per task suite". Below the title is a subtitle: "You can add one or more tasks to the page. Enter the total price for all tasks on the page". Inside the box, there is a label "PRICE IN US DOLLARS" followed by a text input field containing "0.01" and a close button (X). To the right of this is a label "FEE" followed by a text input field containing "0.005". At the bottom of the box is a button labeled "+ Dynamic pricing".

4. Set up user [filters](#).
  - Select English-speaking performers using the "Language = English" filter. Prevent performers who completed previous tasks from checking this one. To do this, set a filter with the ["Found\\_shoes" skill](#):  
*The "Found\_shoes" skill = absent (empty field)*

The screenshot shows a configuration box titled "User filter". At the top right is a link "Copy settings from...". Below the title is a section with a dropdown menu labeled "Add filter" and a button labeled "Create skill". Below this is a section labeled "PROFILE" with a dropdown menu showing "Languages" and a text input field containing "English". Below this is a section labeled "SKILLS" with a dropdown menu showing "found\_shoes" and a text input field containing "Absent". There are also "AND" and "+" buttons between the sections.

Optionally, specify the percentage of top-rated performers in the [Speed / Quality ratio](#).  
**Important:** This can slow down pool completion.

**Speed/quality ratio** ?

Top % Online Time

Specify the percentage of top-rated active users who can access tasks in the pool.

3523 Speed 352 Quality

60% top-rated performers were selected.  
The task is available to 2113 active users.

- Set up [quality control](#):  
[Golden Set aka Control tasks](#). Ban performers who give incorrect responses to control tasks. Example:

**CONTROL TASKS** ?

Recent values to use

If   +

and   ×

then

×

[Fast responses](#). You can ban the performers who suspiciously fast responses. This way you can get rid of cheaters in your pool. Example:

**FAST RESPONSES** ?

Recent values to use

Minimum time per task suite  ×

If   +

then

×

- Overlap. This is the number of users who will complete the same task. For example, 3 is enough for aggregation.

**Overlap**

Specify how many performers you want to complete each task in the pool.

×

DYNAMIC OVERLAP ?

- Time allowed for completing a task page (for example, 300 seconds).
- Keep task order for your convenience

Parameters

TIME FOR COMPLETING A TASK PAGE IN SECONDS.

POOL CLOSING DATE

KEEP TASK ORDER
☒ Yes

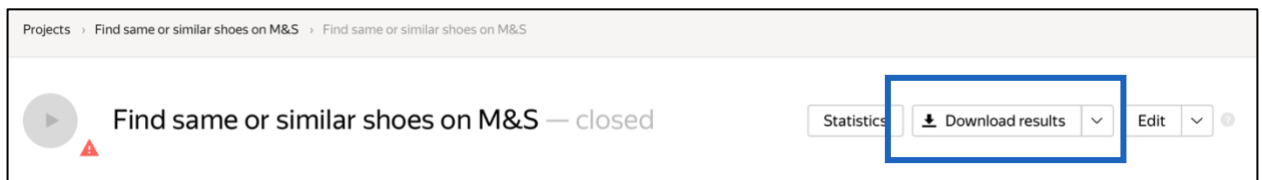
TIME BEFORE POOL CLOSES IN SECONDS

POOL PRIORITY IN PROJECT

9. Save the pool.

## Preparing and uploading a file with tasks

1. Wait until the pool of [project #2 on "finding similar shoes"](#) is completed.
2. Open the pool page in Project #2 and click the **Download results** button



- Clear the **Accepted** checkbox and select **Submitted**.
- Clear **link**, **user ID**, **status**, **start time** and **Separate assignments with empty row** checkboxes. This will give you a list of unreviewed tasks.

Download results

Status
☐ Active
☒ Submitted
☐ Accepted

☐ Rejected
☐ Skipped
☐ Expired

Columns
☐ link
☒ assignment id
☐ task suite ID

☐ user ID
☐ status
☐ start time

☐ submit time
☐ accept time
☐ reject time

☐ skip time
☐ expire time
☐ price

☐ Download data for the period

☐ Separate assignments with empty row

☐ Exclude assignments by banned users

Close

Download results

3. Keep and rename the following columns:
  - Keep the name of the **"INPUT:image"** column as it is.

- Change the name of the "**OUTPUT:found\_link**" column to "**INPUT:found\_link**". To check this image for correctness in project 3.
- Change the "**ASSIGNMENT:assignment\_id**" column name to **INPUT:assignment\_id** to later track and match the assignment number.

**Make sure you have the headers of the columns exactly as below**

A	B	C
INPUT:image	INPUT:found_link	INPUT:assignment_id
https://tlk.s3.ya	https://www.asos.c	https://www.asos.com/a
https://tlk.s3.ya	https://www.asos.c	https://www.asos.com/r
https://tlk.s3.ya	https://www.asos.c	https://www.asos.com/r
https://tlk.s3.ya	https://www.asos.c	https://www.asos.com/r

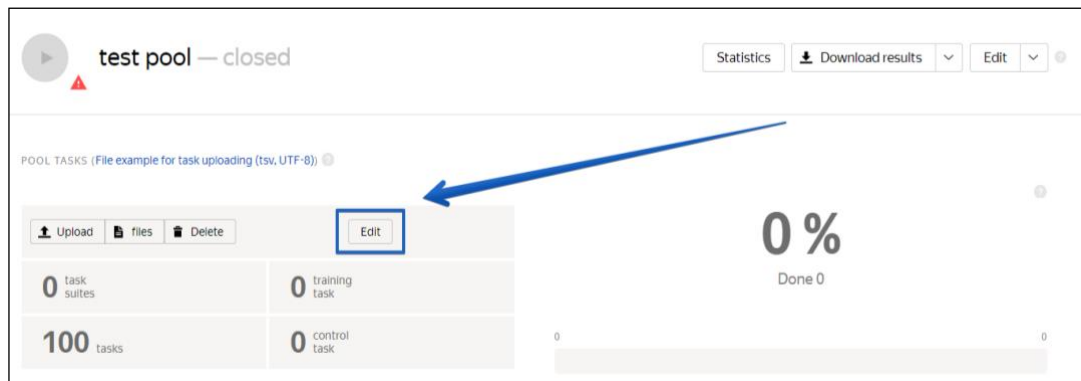
Save the file in TSV format.

4. Open the [pool page in Project #3](#).
5. [Upload the file to the pool](#) by selecting **Smart mixing**. Set the number of main and control tasks per page (for example, 9 and 1).

6. [Create control tasks](#).

**Note.** Control tasks are tasks with the correct response known in advance. They are used to track the performer's quality of responses. The performer's response is compared to the response you provided. If they match, it means the performer answered correctly.

Click **Edit** → **Create control tasks**.



### Edit tasks

Use main tasks as a starting point to create control tasks or training tasks.

Control tasks are for checking the quality of responses from performers. They contain correct responses to compare with actual responses.

Training tasks are for teaching performers how to complete tasks. They contain correct responses and hints.

[Learn more](#)

Main 100   Control tasks 0   Training tasks 0

**Create control tasks**   Create training tasks   Download

ID	Overlap	Responses from performers	Last updated
...1975eb01	3	0	07/26/2019 22:18:06
...1975eabe	3	0	07/26/2019 22:18:06
...1975ead1	3	0	07/26/2019 22:18:06

- Check the "result" output field that is used to match the user response to the control one, select the response and click **Save and go to next**.

Projects > Do the shoes in the picture look... > Do the shoes in the picture look... > Uploaded tasks > Edit tasks

#### Create control task

1. Specify the correct answers

2. Select the fields to include

☒ Field Value

☒ result No

**Save and go to next**

Control tasks: 6

ID	Details	Updated:
...87c98746	Overlap: 3 Main Responses: 3	06/04/2020
...87c98747	Overlap: 3 Main Responses: 3	06/04/2020
...87c98745	Overlap: 3 Main Responses: 3	06/04/2020
...87c98748	Overlap: 3 Main Responses: 3	06/04/2020
...87c98743	Overlap: 3 Main Responses: 3	06/04/2020
...87c98741	Overlap: 3 Main Responses: 3	06/04/2020
...87c98742	Overlap: 3 Main Responses: 3	06/04/2020
...87c98744	Overlap: 3 Main Responses: 3	06/04/2020
...87c98746	Overlap: 3 Main Responses: 3	06/04/2020
...87c98776	Overlap: 3 Main Responses: 3	06/04/2020
...87c98773	Overlap: 3 Main Responses: 3	06/04/2020
...87c98775	Overlap: 3 Main Responses: 3	06/04/2020
...87c98774	Overlap: 3 Main Responses: 3	06/04/2020
...87c98777	Overlap: 3 Main Responses: 3	06/04/2020

Enter correct responses for your control tasks. In small pools, control tasks should account for approximately 10% of all tasks.



**Tip.** Make sure to include different variations of correct responses in equal amounts.  
Open the **Control tasks** → **Distribution of correct responses for control tasks** tab.

Projects > Does the image contains t... > test pool > Uploaded tasks

### Edit tasks

Use main tasks as a starting point to create control tasks or training tasks.  
Control tasks are for checking the quality of responses from performers. They contain correct responses to compare with actual responses.  
Training tasks are for teaching performers how to complete tasks. They contain correct responses and hints.  
[Learn more](#)

Main 90 **Control tasks 10** Training tasks 0

Create control tasks Create training tasks Download

ID	Overlap	Responses from performers	Last updated
...19f9102c	3	0	07/26/2019 21:45:27
...19f9101b	3	0	07/26/2019 21:45:27

Main 90 **Control tasks 10** Training tasks 0

Create from main tasks Download

ID	Overlap	Responses from performers	Correct responses, %	Last updated
...19f90ff6	∞	0		07/26/2019 21:45:27
...19f90ff0	∞	0		07/26/2019 21:45:27
...19f90ff1	∞	0		07/26/2019 21:45:27

**Distribution of correct responses for control tasks**

result

40% BAD

60% OK

- Save the markup and check the number of control tasks.

test pool — closed

Statistics Download results Edit

POOL TASKS (File example for task uploading (tsv, UTF-8))

Upload files Delete Edit Preview

~30 task suites 0 training task

90 tasks **10 control task**

0 % Done 0

7. Start the pool.

**Important.** Remember that real Toloka performers will complete the tasks. Double check that everything is correct with configuration of your project before you start the pool.

test pool — closed

Statistics Download results Edit

POOL TASKS (File example for task uploading (tsv, UTF-8))

Upload files Delete Edit Preview

~30 task suites 0 training task

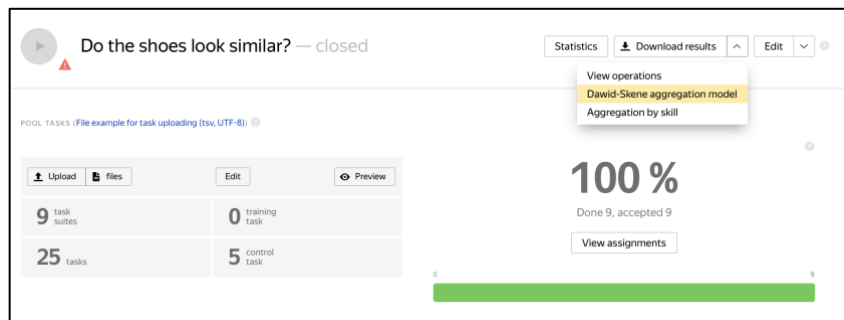
90 tasks 10 control task

0 % Done 0

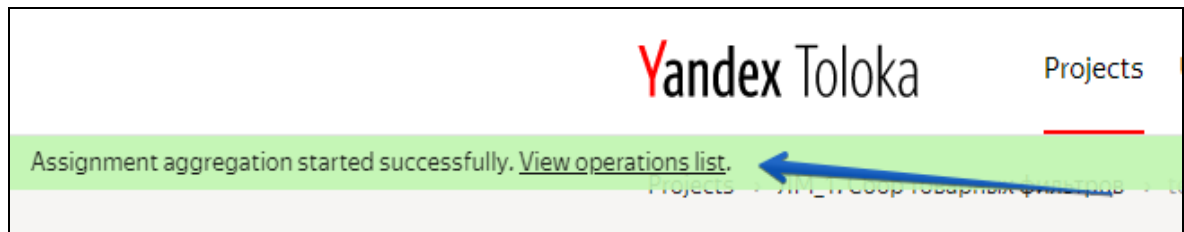
## Receiving responses

**Disclaimer: Aggregation takes from 5 to 20 minutes. During this time, you can start configuring your next project. Refresh the Operations page to check progress.**

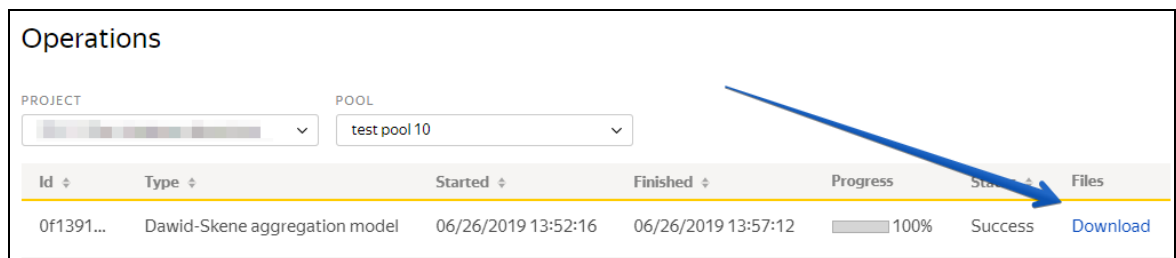
1. Wait until the pool is completed.
2. Click the arrow next to the **Download results** button and run aggregation using the [Dawid-Skene model](#).



3. Go to the operations list and wait until aggregation finishes.



4. Download the responses.



## Upload reviewed results

As you set **post verification** in the [pool settings in Project #2](#), you need to check the performers' responses within the time limit set in the **Deadline** field.

1. Open the [file with aggregated results](#) in a spreadsheet editor.
2. Add the following columns:
  - **"ACCEPT:verdict"** — The result of verification.
  - **"ACCEPT:comment"** — Comments for performers if responses were rejected (for example, which part of the instructions wasn't followed).
3. Change the name of the **"INPUT:assignment\_id"** column to **"ASSIGNMENT:assignment\_id"**.
4. Delete all other columns
5. Fill in the **"ACCEPT:verdict"** and **"ACCEPT:comment"** columns:
  - If the aggregated result for the task is OK, put **"+"** then the task will be accepted.
  - If the result is BAD or 404, put **"-"** then the task will be rejected. Enter the reason for rejection in the **"ACCEPT:comment"** field . For example: *The item provided is incorrect or improper.*
6. Now you can delete the other columns. Save the edited TSV file.

A	B	C
ASSIGNMENT:assignment_id	ACCEPT:verdict	ACCEPT:comment
00009c4245--5e27381d139baa	+	
00009c4245--5e27337bb0e86f	-	The item provided is incorrect
00009c4245--5e2737692f3cb8	+	
00009c4245--5e2739bcb0e86f	+	

7. Open the [pool page in Project #2](#).
8. Click [Review assignments](#) on the pool page above the progress bar.
9. Click **Upload review results**.

Submitted responses

Download results

Upload review results

30 All assignments	30 Under review	0 Accepted assignments	0 Rejected assignments
-----------------------	--------------------	---------------------------	---------------------------

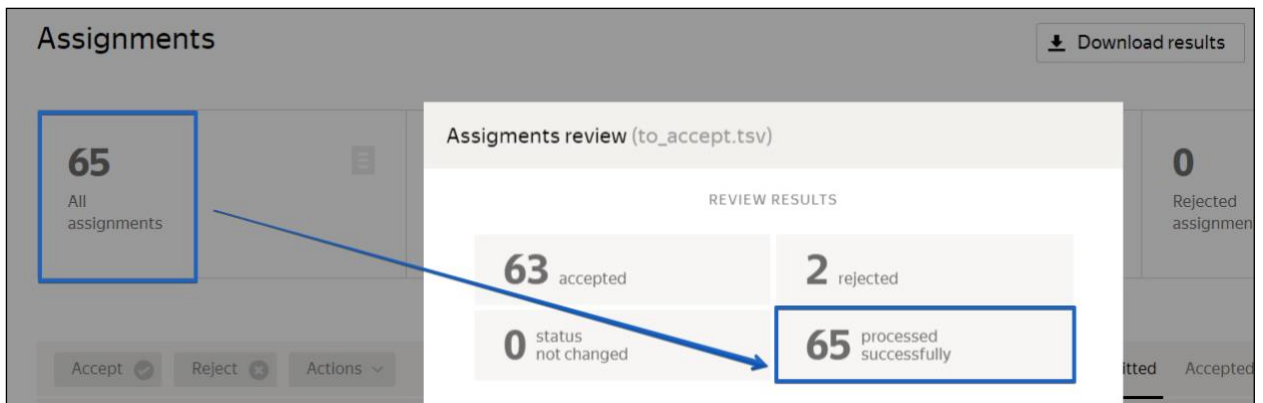
Accept

Reject

Actions

				Submitted	Accepted	Rejected	All assignments
Response	User	Completed	Duration				Status
00009e68cb--5e2dd577f0fb3d0120662698	483927a3f25b15f6a85c7f97d2a1c9f5	01/26/2020 21:08:57	1 min 5 sec	—			
00009e68cb--5e2dd5bae5ae53011fc20a19	483927a3f25b15f6a85c7f97d2a1c9f5	01/26/2020 21:09:36	38 sec	—			
00009e68cb--5e2dd5e1d52a900123d8dcde	483927a3f25b15f6a85c7f97d2a1c9f5	01/26/2020 21:10:53	1 min 15 sec	—			

10. Select the file and upload it to Toloka.
11. Check that all tasks have changed their status to accepted or rejected.



12. You rejected tasks and set up the rule to send them for re-completion while configuring Project 2.

**If have enough time you can do as many more reiterations as needed in order to receive as much clean data as you can.** The steps are the following: The pool will open again, and these tasks will be resent to other performers. After the pool is marked up, download the new results and submit them for review. Download the reviewed results. Repeat these steps until all the images from the second project are correctly marked up.

**But if you do not have enough time, move on to the next project and you can complete the reiterations later at your own pace.**

## Review assignments in the interface (another way of results validation)

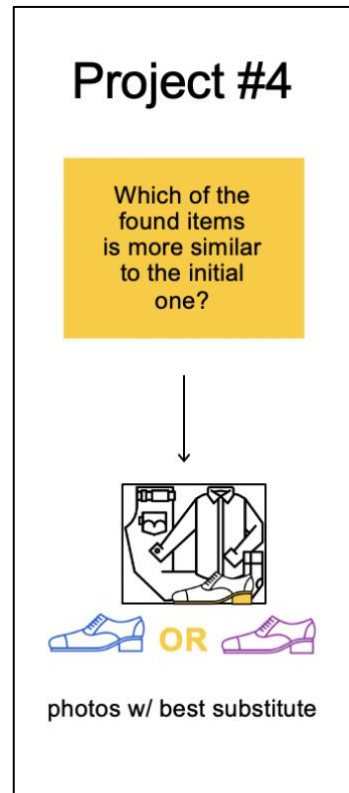
You can also [review assignments by yourself](#) and see the results of the crowdsourcing pipeline that you have created.

1. Open the [pool page in Project #2](#).
2. Click the **Review assignments** button on the pool page.

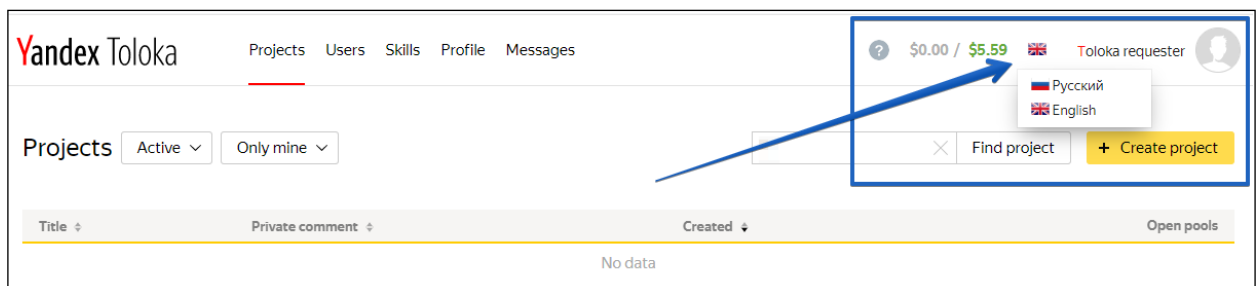
Assignment	User	Date	Durat.	Submitted	Accepted	Rejected	All assignments	Status
<input type="checkbox"/> 0000586e7e--5d2c9c446064f701220ca9c5	788168bc6cebfbbaade32a38035e505b9	07/15/2019 18:32:06	49 sec	—				●
<input type="checkbox"/> 0000586e7e--5d2c9c3ab5cff0011e313fa0	e85a361c96450663de00c64a12d9385f	07/15/2019 18:32:28	1 min 21 sec	—				●

- Choose an assignment then click **Accept** or **Reject**.
- For rejected assignments, enter a comment (explain why you decline it).

# Project #4 Which item is more similar?

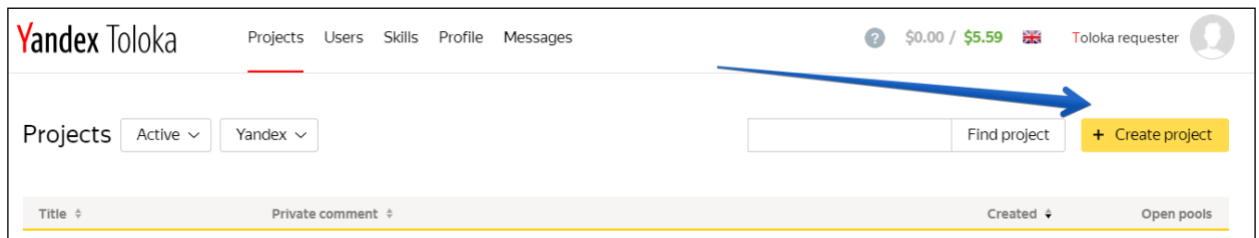


**Important:** Before you start using Toloka, make sure that the **English** language is selected.

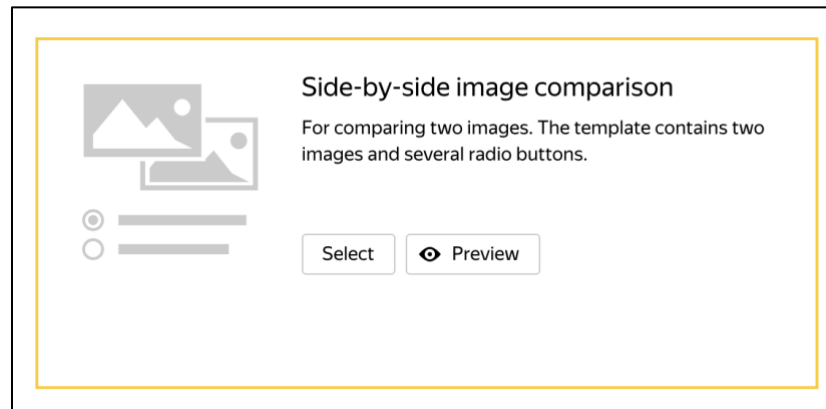


## Project creation

1. Click the button **+ Create project**



2. Choose the **Side-by-side image comparison** template.



1. Enter a clear project name and description.  
**Important:** It will be visible for real Toloka performers.
2. Write short and simple instructions. Example:

A screenshot of the project creation form in Yandex Toloka. It has three main sections: 'PROJECT NAME', 'DESCRIPTION', and 'INSTRUCTIONS'. The 'PROJECT NAME' field contains the text 'Which shoes look more similar?'. The 'DESCRIPTION' field contains the text 'Decide which pair of shoes look more alike to the initial one'. The 'INSTRUCTIONS' field has a rich text editor with a toolbar. The text inside the editor reads: 'What you need to do:  
  
Look at 2 pictures with different shoes and decide which pair of shoes look more similar to the initial pair. Use your own sense of style, but also remember that they will look alike if they are similar color, similar form, similar fabric and similar length :)  
  
Good luck!'

11. Define parameters for the **input and output data**:

- The **"image"** field is the initial image from the dataset.
- The **"left\_link"** field is a link that the performer provided to match the item from the initial image.
- The **"right\_link"** field is another link the performer provided.

*You will be able to upload the file with links to the pool later.*

- The **"result"** field will be used to receive performer's responses.

The code for specifications is:

Input data:

```
{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "left_link": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "right_link": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}
```

Output data:

```
{
  "result": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}
```

## 12. Create the task interface in the HTML block.

```
<div class="header">
  <div class="left caption">
    {{button label="Go to site" href=uploaded_link_left size="L"}}
    <p class="url">{{uploaded_link_left}}</p>
  </div>
  <div class="right caption">
    <p class="url">{{uploaded_link_right}}</p>
    {{button label="Go to site" href=uploaded_link_right size="L"}}
  </div>
</div>

{{img src=image}}

<div class="content clearfix">
  <div class="left page">
    {{iframe src=uploaded_link_left width="100%" height="700px" real-
size=true screenshot=true}}
  </div>
  <div class="right page">
    {{iframe src=uploaded_link_right width="100%" height="700px" real-
size=true screenshot=true}}
  </div>
</div>

<div class="footer">
  {{field type="radio" name="result" label="The left one is better"
value=result_left hotkey="1"}}
  {{field type="radio" name="result" label="The right one is better"
value=result_right hotkey="2"}}
</div>
```

13. **DO NOT delete any of JS code** but add the following JS block right before [OnRender](#)

```
getTemplateData: function() {
    var data = TolokaHandlebarsTask.prototype.getTemplateData.apply(this,
arguments),
    input = this.getTask().input_values;
    var left_link = input.left_link;
    var right_link = input.right_link;
    var uploaded_link_left = '',
        uploaded_link_right = ''
    if (Math.floor(Math.random() * 2)) {
        uploaded_link_left = left_link
        uploaded_link_right = right_link
    } else {
        uploaded_link_left = right_link
        uploaded_link_right = left_link
    }
    data.uploaded_link_left = uploaded_link_left;
    data.uploaded_link_right = uploaded_link_right;
    data.result_left = uploaded_link_left;
    data.result_right = uploaded_link_right;

    return data;

},
```

14. In the CSS block add:

```
.task {
    display: block;
    text-align:center;
}

.header {
    overflow: hidden;
    background-color: #FFCC00;
}

.caption {
    width: 50%;
}

.url {
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;

    max-width: calc(100% - 182px);

    display: inline-block;
    vertical-align: bottom;
}

.button {
    margin: 10px;
    max-width: 182px;
}

.button__label {
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    max-width: 150px;
}

.content {
```



```

margin: 10px 0;
}

.page {
display: inline-block;
width: 50%;
}

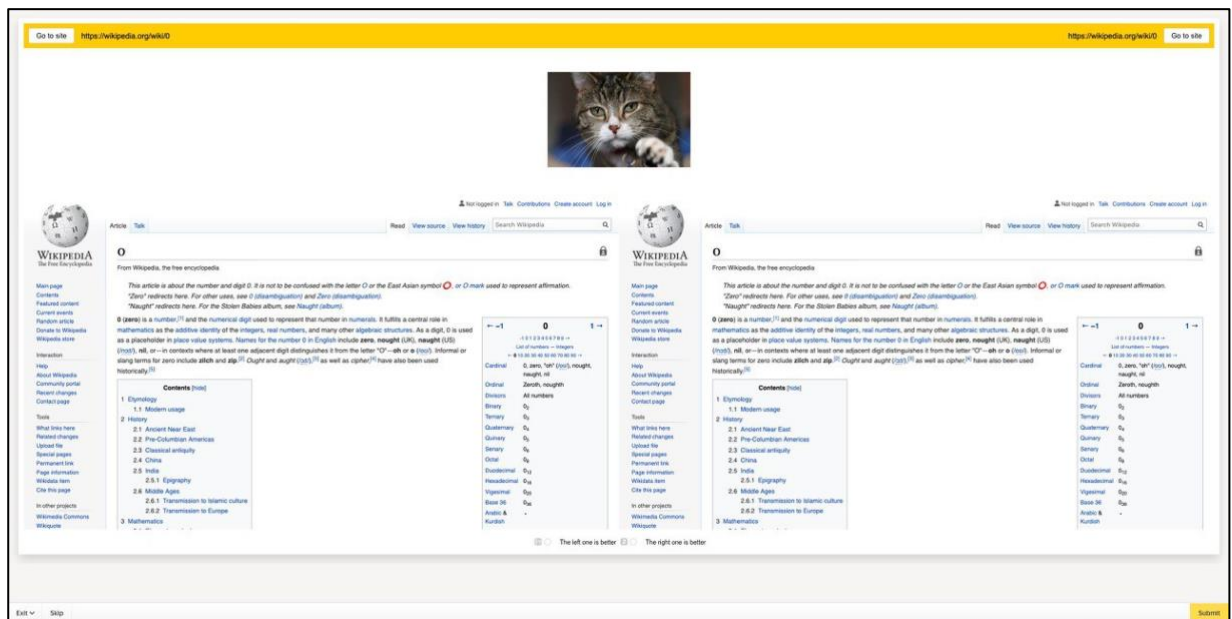
.left {
float: left;
text-align: left;
}

.right {
float: right;
text-align: right;
}

.clearfix {
overflow: hidden;
width: 100%;
}

```

15. Click the **Preview** button to see the performer's view of the task.  
*You will see standard pictures on the page.*

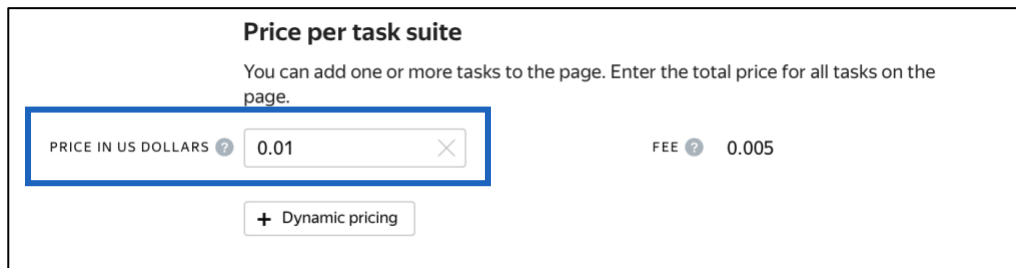


16. Select the radio buttons in the preview and make sure that the task can be completed.  
 17. Click **Save** button to save the project.

**Note.** To edit project parameters, click the button in the list of projects or **Project actions** → **Edit** on the project page.

## Pool creation

1. Click **Add pool**.
2. Give the pool any convenient name and description. You are the only one who can see them.
3. Specify the [pool parameters](#):
  - Price per task page (for example, \$0.01)



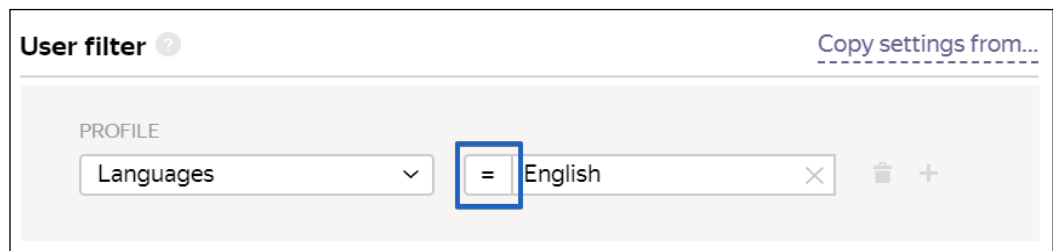
**Price per task suite**

You can add one or more tasks to the page. Enter the total price for all tasks on the page.

PRICE IN US DOLLARS

FEE

4. Set up user [filters](#).
  - Select English-speaking performers using the **Language = English** filter.



**User filter**

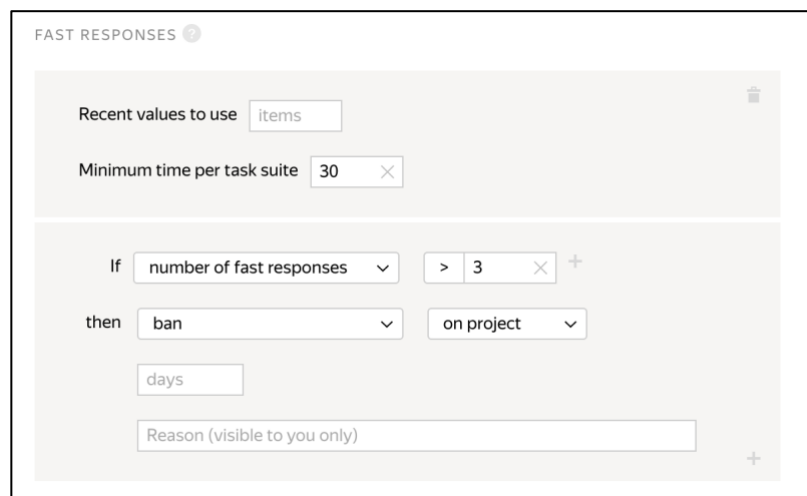
[Copy settings from...](#)

PROFILE

Languages   English

5. Set up [quality control](#):

[Fast responses](#). You can ban the performers who suspiciously fast responses. This way you can get rid of cheaters in your pool. Example:



**FAST RESPONSES**

Recent values to use

Minimum time per task suite

If

then

Optionally, add [other quality control rules](#).

6. Overlap. This is the number of users who will complete the same task. Put a larger number in this task. **For example, 10.**

### Overlap

Specify how many performers you want to complete each task in the pool.

OVERLAP ?

10

×

DYNAMIC OVERLAP ? ☐ Off

7. Optionally, specify the percentage of top-rated performers in the [Speed / Quality ratio](#). **Important:** This can slow down pool completion.

### Speed/quality ratio ?

Top %

Online

Time

Specify the percentage of top-rated active users who can access tasks in the pool.

3523

Speed

All

90%

80%

70%

60%

50%

40%

30%

20%

10%

Quality

352

**60%** top-rated performers were selected.  
The task is available to **2113** active users.

8. Time allowed for completing a task page (for example, 300 seconds).

### Parameters

TIME FOR COMPLETING A TASK PAGE IN SECONDS. ?

300

×

POOL CLOSING DATE ?

2021-06-04

KEEP TASK ORDER ?

Yes

TIME BEFORE POOL CLOSES IN SECONDS ?

0

POOL PRIORITY IN PROJECT ?

0

9. Save the pool.

## Preparing and uploading a file with tasks

1. Take the downloaded TSV file with validated responses from Project 3.
2. Now you need to generate pairs for each INPUT:image so that you will be able to compare two found images with the initial one and decide which one is more similar than another.

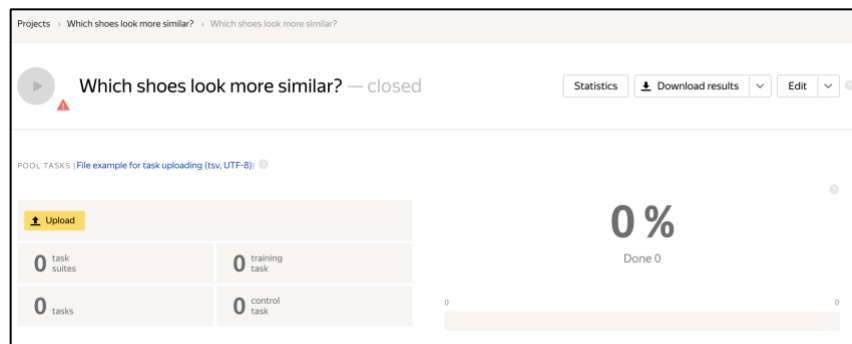
**You can either generate the pairs by hand, using MS Excel or any other editor or you can automatically do it. We recommend using Python and Jupyter Lab.**

You can consult with our results

[https://tlk.s3.yandex.net/wsdm2020/SbS\\_Toloka\\_prep&aggr\\_data.ipynb](https://tlk.s3.yandex.net/wsdm2020/SbS_Toloka_prep&aggr_data.ipynb)

3. [Upload pool tasks](#) from this file.

**Important:** If you changed the name of the input field, change it in the file as well



[Upload the file](#) to the pool by selecting **Set manually** and specify the number of tasks per page.

**You can experiment with the number of tasks!**

4. Start the pool.  
**Important.** Remember that real Toloka performers will complete the tasks. Double check that everything is correct with configuration of your project before you start the pool.

## Receiving responses

1. Wait until the pool is completed. Refresh the pool page to check progress.
2. Download **the accepted** results. Select the URLs, user IDs and assignment IDs like in the picture below. **Do not forget to clear “Separate assignments with empty row” box!**

Download results

Status

☐ Active ☐ Submitted ☒ Accepted

☐ Rejected ☐ Skipped ☐ Expired

Columns

☒ URL ☒ assignment ID ☐ task suite ID

☒ user ID ☐ status ☐ start time

☐ submit time ☐ accept time ☐ reject time

☐ skip time ☐ expire time ☐ price

☐ Download data for the period

☐ Separate assignments with empty row

☐ Exclude assignments by banned users

Close Download results

3. Try to run Bradley Terry model on these results (you can consult with our results [https://tlk.s3.yandex.net/wsdm2020/SbS\\_Toloka\\_prep&aggr\\_data.ipynb](https://tlk.s3.yandex.net/wsdm2020/SbS_Toloka_prep&aggr_data.ipynb))

**GOOD LUCK!!!**

# Appendix: Expanded code of the projects

## Project 1

### Specifications:

#### Input:

```
{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}
```

#### Output:

```
{
  "result": {
    "type": "string",
    "hidden": false,
    "required": true
  }
}
```

### HTML:

```
{{img src=image width="100%" height="400px"}}
<div>Are there <b>shoes</b> in the picture?</div>
<div> {{field type="radio" name="result" value="Yes" label="Yes"
hotkey="1"}} {{field type="radio" name="result" value="No" label="No"
hotkey="2"}}</div>
```

### JS:

```
exports.Task = extend(TolokaHandlebarsTask, function (options) {
  TolokaHandlebarsTask.call(this, options);
}, {
  onRender: function() {
    // DOM element for task is formed (available via #getDOMElement())
  },
  onDestroy: function() {
    // Task is completed. Global resources can be released (if used)
  }
});

function extend(ParentClass, constructorFunction, prototypeHash) {
  constructorFunction = constructorFunction || function () {};
  prototypeHash = prototypeHash || {};
  if (ParentClass) {
    constructorFunction.prototype = Object.create(ParentClass.prototype);
  }
  for (var i in prototypeHash) {
```

```

    constructorFunction.prototype[i] = prototypeHash[i];
  }
  return constructorFunction;
}

```

## Project 2

### Specifications:

#### Input:

```

{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}

```

#### Output:

```

{
  "button": {
    "type": "boolean",
    "hidden": false,
    "required": true,
    "allowed_values": [
      true
    ]
  },
  "found_link": {
    "type": "string",
    "hidden": false,
    "pattern":
      "https://www.marksandspencer.com/.*",
    "required": true
  },
  "found_image": {
    "type": "file",
    "hidden": false,
    "required": true
  }
}

```

#### HTML:

```

{{img src=image width="50%" height="400px"}}
<div class='answers'>
  <p>Find the same <b>shoes</b> on Marks and Spencer</p> {{field
type="button-clicked" name="button" label="Marks and Spencer"
href="https://www.marksandspencer.com" action=true}}
  <p>Shoes must be the same color and the same style.</p>
  <p>Paste the link here</p> {{field width="100%" type="input"
name="found_link"}}
  <p>Upload the image here</p>
  <div> {{field width="100%" type="file-img" name="found_image"
preview=true}} </div>
</div>

```

#### JS:

```

exports.Task = extend(TolokaHandlebarsTask, function (options) {
  TolokaHandlebarsTask.call(this, options);
}, {
  onRender: function() {
    // DOM element for task is formed (available via #getDOMElement())
  },
  onDestroy: function() {
    // Task is completed. Global resources can be released (if used)
  }
});

function extend(ParentClass, constructorFunction, prototypeHash) {
  constructorFunction = constructorFunction || function () {};
  prototypeHash = prototypeHash || {};
  if (ParentClass) {
    constructorFunction.prototype = Object.create(ParentClass.prototype);
  }
  for (var i in prototypeHash) {
    constructorFunction.prototype[i] = prototypeHash[i];
  }
  return constructorFunction;
}

```

### **CSS:**

```

.task {
  display: block;
  height: 500px;
  width: 800px;
}
.img {
  float: left;
  width: 50%;
}
.answers {
  float: left;
  width: 40%;
  margin: 5%;
}

```

## **Project 3**

### **Specifications:**

#### **Input:**

```

{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "found_link": {
    "type": "url",
    "hidden": false,
    "required": true
  },

```

#### **Output:**

```

{
  "result": {
    "type": "string",
    "hidden": false,
    "required": true
  }
}

```



```

    "assignment_id": {
      "type": "string",
      "hidden": true,
      "required": true
    }
  }
}

```

### HTML:

```

{{img src=image height="400px"}} {{iframe src= found_link height="600px"}}
<p>Check that the uploaded image matches the product in the store.</p>
{{button label="Check the item" href=found_link action=true}}
<p>Are these <b>shoes</b> similar to each other?</p>
<p>Shoes must be the same color and the same style.</p>
{{field type="radio" name="result" value="Yes" label="Yes"}}
{{field type="radio" name="result" value="No" label="No"}}

```

### JS:

```

exports.Task = extend(TolokaHandlebarsTask, function (options) {
  TolokaHandlebarsTask.call(this, options);
}, {
  onRender: function() {
    // DOM element for task is formed (available via #getDOMElement())
  },
  onDestroy: function() {
    // Task is completed. Global resources can be released (if used)
  }
});

function extend(ParentClass, constructorFunction, prototypeHash) {
  constructorFunction = constructorFunction || function () {};
  prototypeHash = prototypeHash || {};
  if (ParentClass) {
    constructorFunction.prototype = Object.create(ParentClass.prototype);
  }
  for (var i in prototypeHash) {
    constructorFunction.prototype[i] = prototypeHash[i];
  }
  return constructorFunction;
}

```

### CSS:

```

.task {
  display: block;
  min-height: 620px;
  width: 100%;
  box-sizing: border-box;
  width: calc(100% - 100px);
}

```

```
.img {
float: left;
width: 30%;
}

.iframe {
float: left;
width: 48%;
margin-left: 10px;
}

.text {
float: left;
width: 18%;
margin-left: 10px;
}
```

## Project 4:

### Specifications:

#### Input:

```
{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "left_link": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "right_link": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}
```

#### Output:

```
{
  "result": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}
```

### HTML:

```
<div class="header">
  <div class="left caption"> {{button label="Go to site"
href=uploaded_link_left size="L"}}
    <p class="url">{{uploaded_link_left}}</p>
  </div>
  <div class="right caption">
    <p class="url">{{uploaded_link_right}}</p>
    {{button label="Go to site" href=uploaded_link_right
size="L"}}
  </div>
</div> {{img src=image}}
<div class="content clearfix">
  <div class="left page">
```

```

        {{iframe src=uploaded_link_left width="100%" height="700px"
real-size=true screenshot=true}}
    </div>
    <div class="right page">
        {{iframe src=uploaded_link_right width="100%" height="700px"
real-size=true screenshot=true}}
    </div>
</div>
<div class="footer">
    {{field type="radio" name="result" label="The left one is better"
value=result_left hotkey="1"}}
    {{field type="radio" name="result" label="The right one is better"
value=result_right hotkey="2"}}
</div>

```

## JS:

```

exports.Task = extend(TolokaHandlebarsTask, function(options) {
    TolokaHandlebarsTask.call(this, options);
}, {
    getTemplateData: function() {
        var data =
TolokaHandlebarsTask.prototype.getTemplateData.apply(this, arguments),
        input = this.getTask().input_values;
        var left_link = input.left_link;
        var right_link = input.right_link;
        var uploaded_link_left = '',
            uploaded_link_right = '';
        if (Math.floor(Math.random() * 2)) {
            uploaded_link_left = left_link
            uploaded_link_right = right_link
        } else {
            uploaded_link_left = right_link
            uploaded_link_right = left_link
        }
        data.uploaded_link_left = uploaded_link_left;
        data.uploaded_link_right = uploaded_link_right;
        data.result_left = uploaded_link_left;
        data.result_right = uploaded_link_right;

        return data;
    },

    onRender: function() {
        // DOM element for task is formed (available via #getDOMElement())
    },
    onDestroy: function() {
        // Task is completed. Global resources can be released (if used)
    }
});

function extend(ParentClass, constructorFunction, prototypeHash) {
    constructorFunction = constructorFunction || function() {};
    prototypeHash = prototypeHash || {};
    if (ParentClass) {
        constructorFunction.prototype =
Object.create(ParentClass.prototype);
    }
    for (var i in prototypeHash) {

```

```

        constructorFunction.prototype[i] = prototypeHash[i];
    }
    return constructorFunction;
}

```

## **CSS:**

```

.task {
    display: block;
    text-align: center;
}

.header {
    overflow: hidden;
    background-color: #FFCC00;
}

.caption {
    width: 50%;
}

.url {
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;

    max-width: calc(100% - 182px);

    display: inline-block;
    vertical-align: bottom;
}

.button {
    margin: 10px;
    max-width: 182px;
}

.content {
    margin: 10px 0;
}

.page {
    display: inline-block;
    width: 50%;
}

.left {
    float: left;
    text-align: left;
}

.right {
    float: right;
    text-align: right;
}

.clearfix {
    overflow: hidden;
}

```

```
width: 100%;  
}
```

```
.image {  
  display: inline-block;  
  width: 50%;  
}
```