

Exercise 4

To fulfill Exercise 4, we chose the domain hotel booking.

We wrote two wrappers: extracting data from **expedia.de** using **import.io** and extracting data from **de.hotels.com** using **Mozenda**.

First we show the process of extracting data:

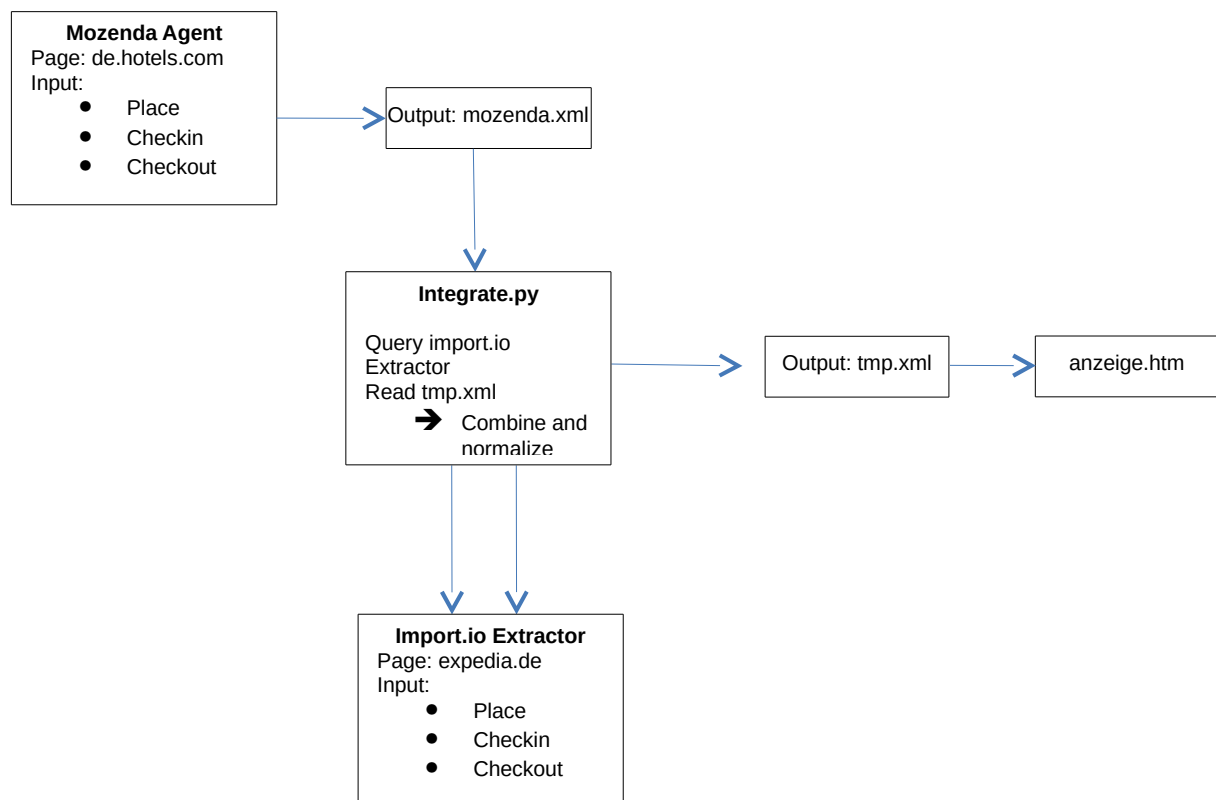


Fig 1: Process of wrapping data from two sources. First, the Mozenda Agent runs manually (because the REST API is not available). Then data are exported as XML and stored in mozenda.xml. The python script integrate.py triggers the import.io extractor with provided user input. So links to detail pages are extracted using one import.io extractor, and these links are used with a second extractor to extract detail hotel information as XML. Data from mozenda.xml is read, and data from both sources are normalized and stored in tmp.xml. anzeige.htm reads the file tmp.xml and displays output in a table. If available, images are displayed, if not, just a text is displayed. The color code corresponds to the wrapper source, after moving the mouse over an hotel thumbnail, data extracted from the detail links is displayed.

The output (anzeige.htm) looks as follows:

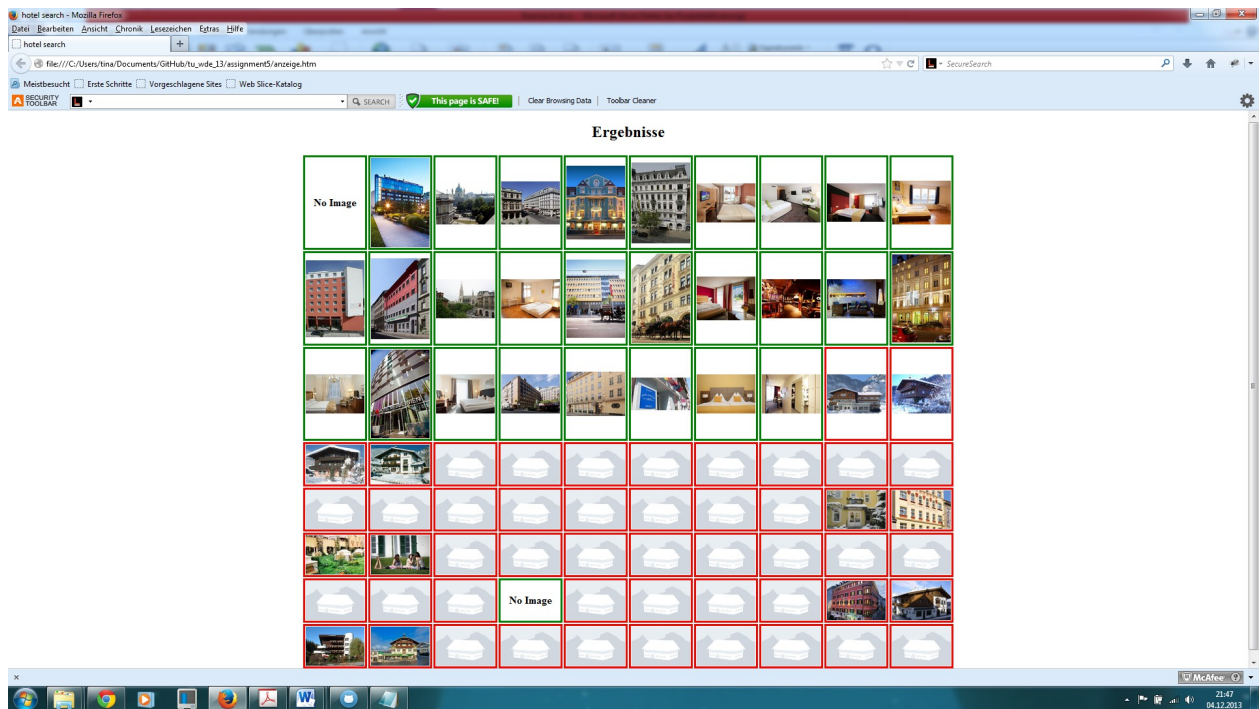


Fig 1: Hotel thumbnails are displayed in a table (if available, otherwise just a text is displayed). A green border corresponds to source mozenda (de.hotels.com), a red border to source import.io (expedia.de).

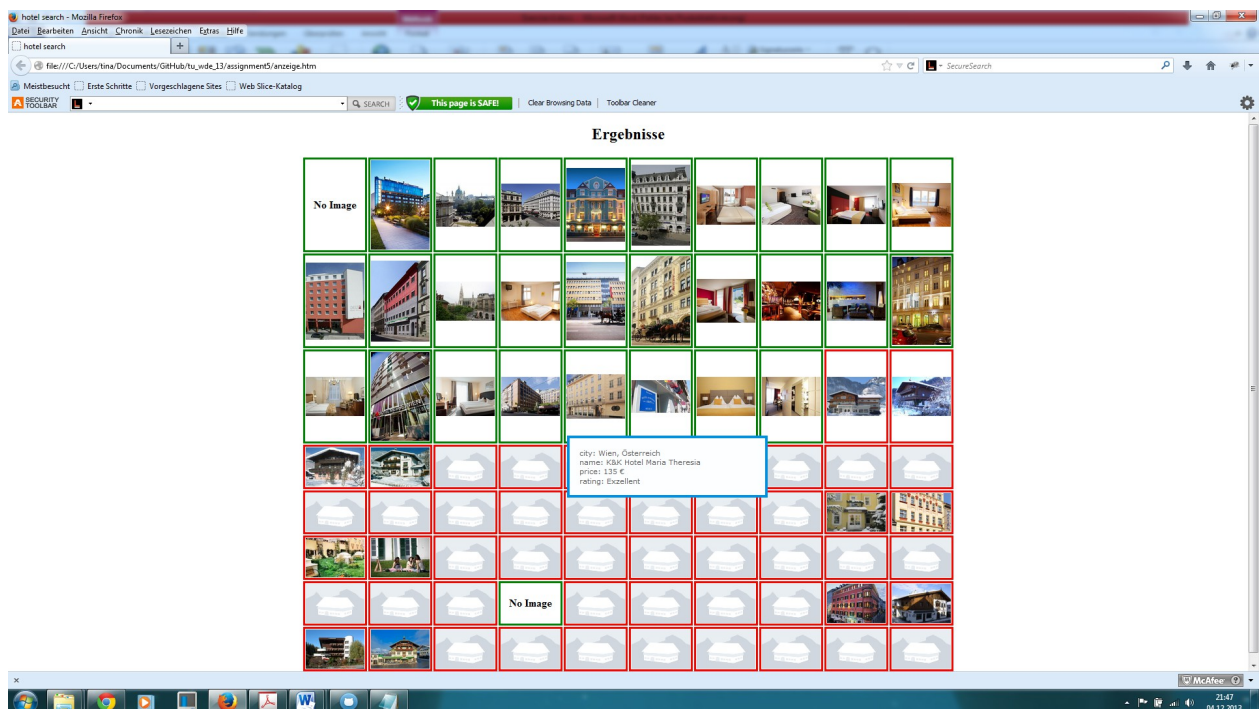


Fig 2: When moving over a hotel thumbnail, information extracted from the detail pages is provided.

Now we provide some information on the used wrappers.

Import.io is a completely web-based builder, very fast to learn, but somehow limited. Pagination should work, you can also train it, but after building the wrapper, it is not carried out. After creating the extractor, we query it using a provided python modul.

Here are screenshots from the work with **import.io**:

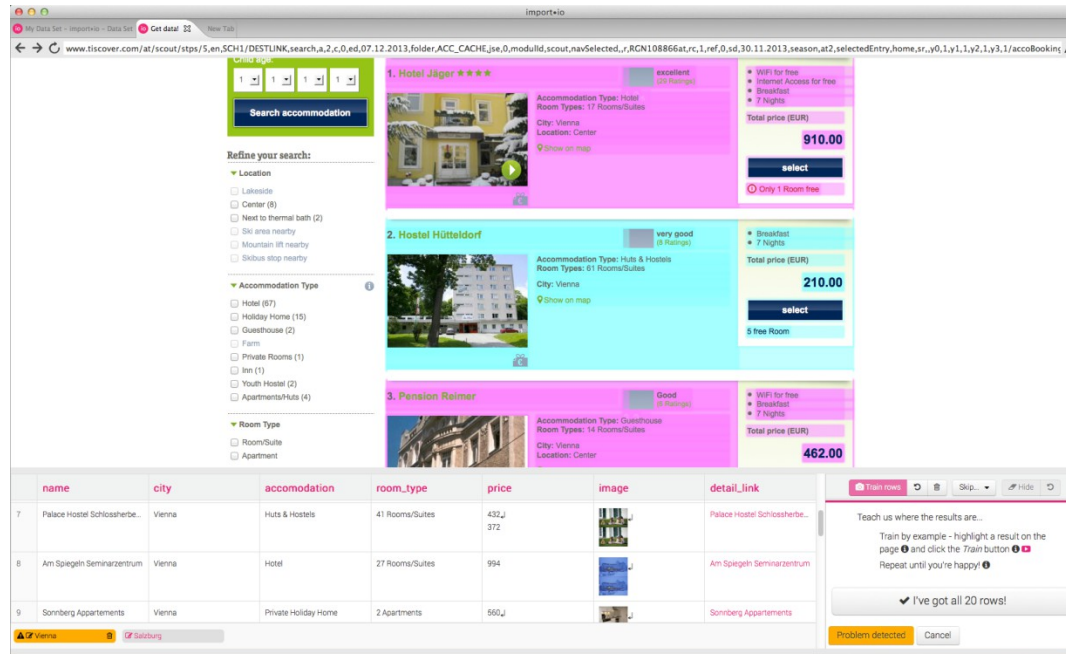


Fig. 4: You always have to choose two to five example objects, so import.io can detect a list of other objects to extract (training the extractor).

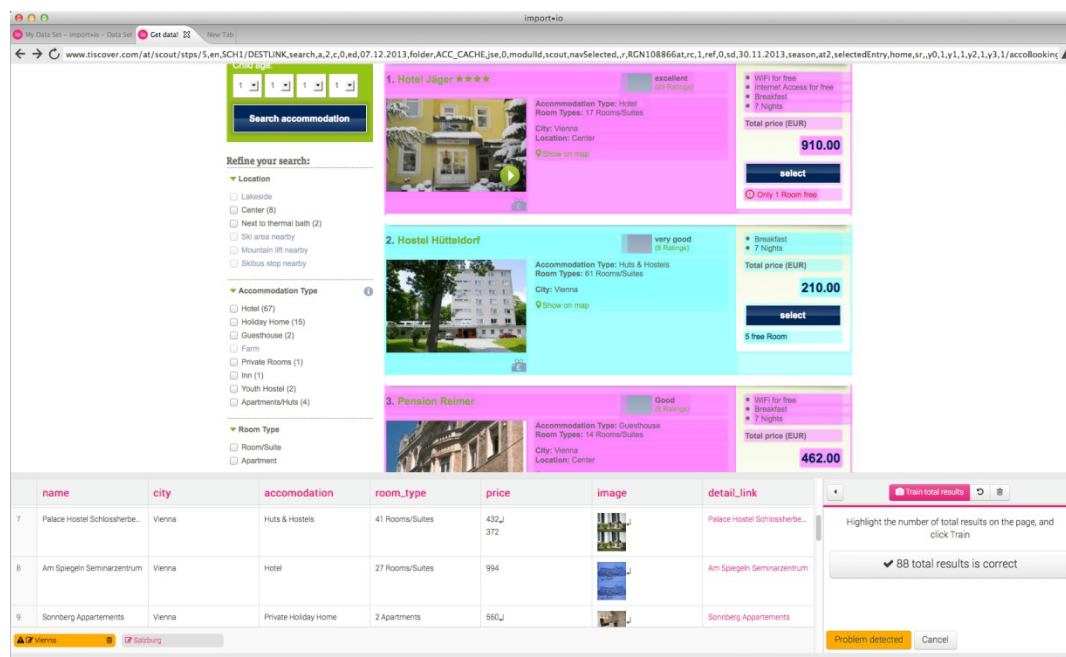


Fig 5: All the captured content is displayed at the bottom for control purpose.

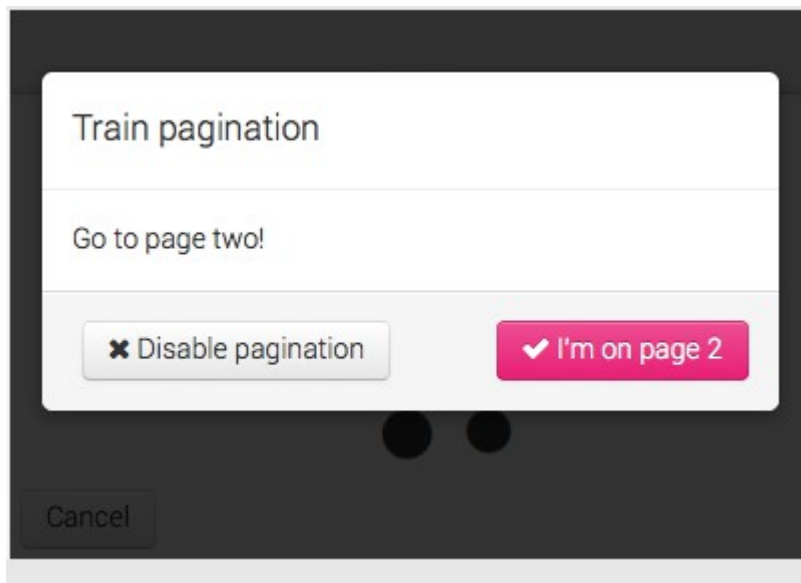


Fig 6: Import.io should be able to train pagination (following next links), but in fact it doesn't work.

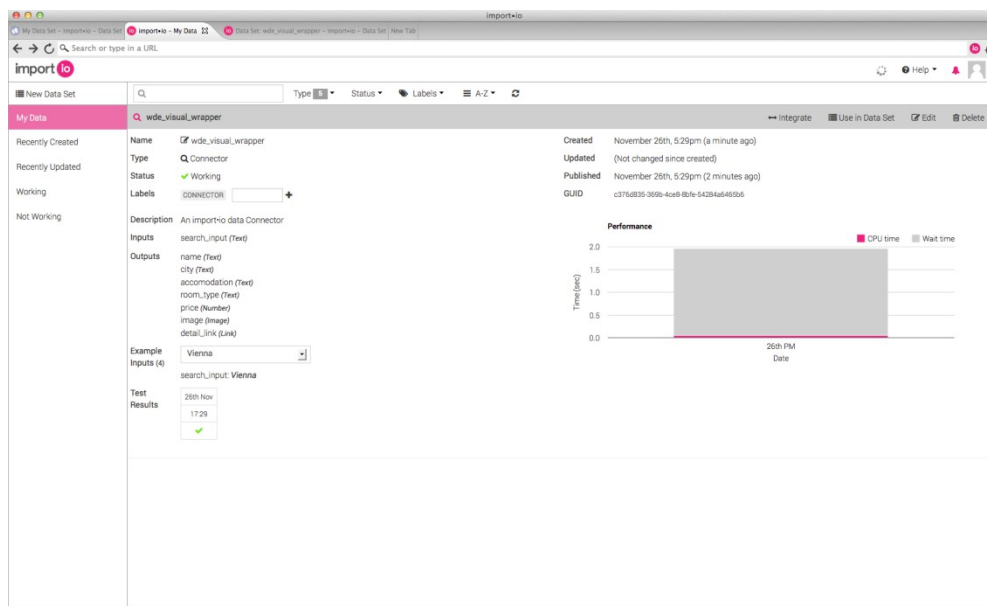


Fig 7: After creating the wrapper, it is stored online and can be queried.

Mozenda is a very easy and intuitive tool for creating web extractors. You have to download a software called AgentBuilder, to build and test your agents locally. After creating an extractor, you upload it to the mozenda server, where it can be run (you can define schedules and query data using the REST API, but it is not available in the trial version). The agent itself is clearly structured, the actions are shown on the left bottom, jumps to other steps (like pagination or following detail links) are illustrated by numbers, and details about every action is displayed in the left upper side.

Here are screenshots from working with **Mozenda**:

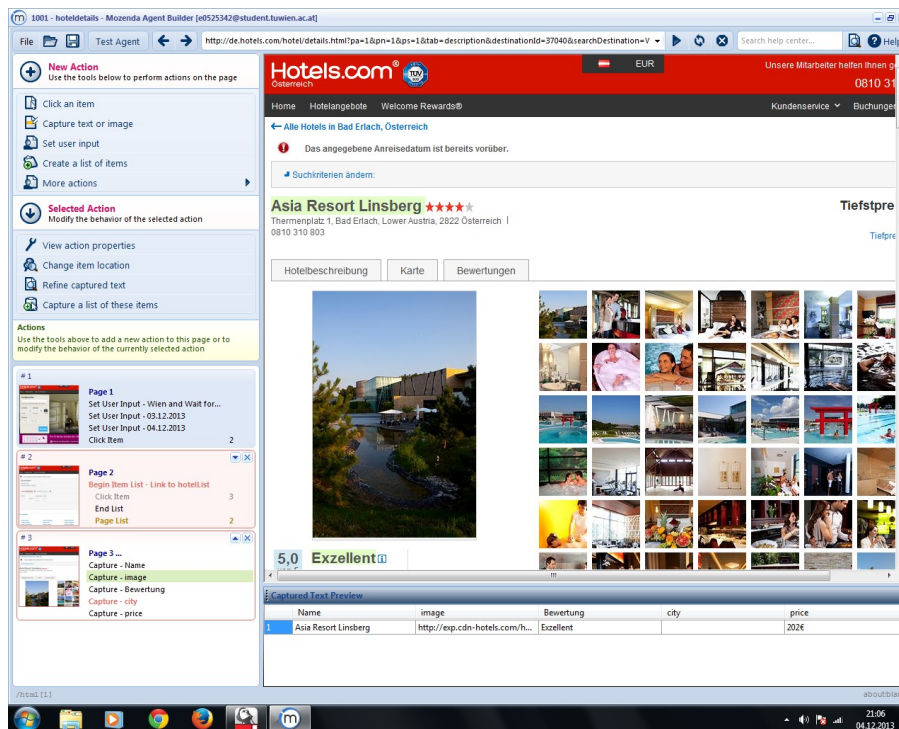


Fig 8: Mozenda is very clearly structured and intuitive; Results are displayed immediately

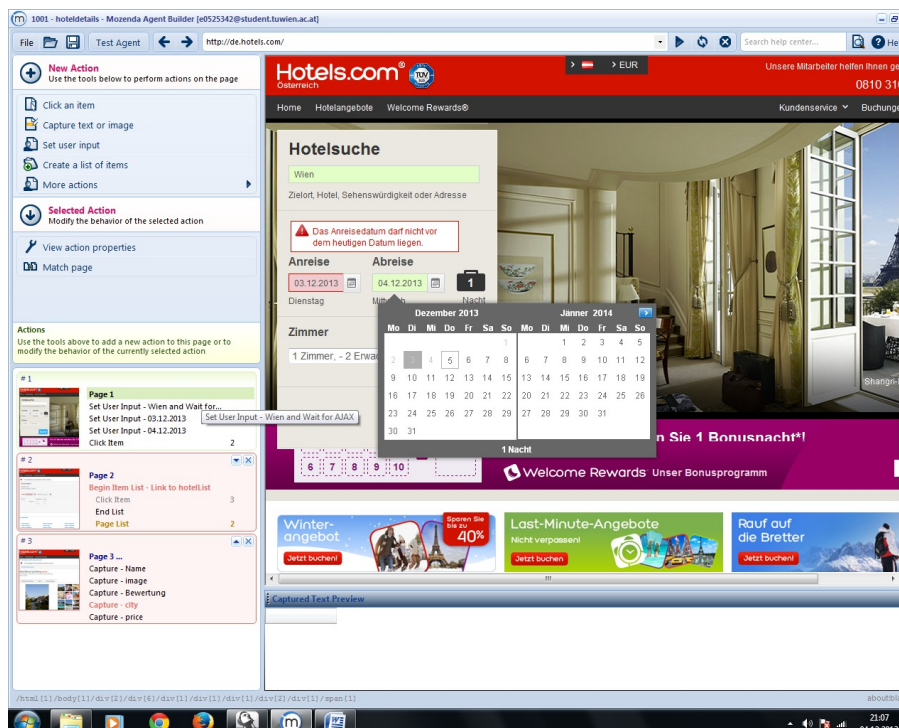


Fig 9: on the left side, you can see the sequence of actions carried out, also the jumps to different steps (like following next or detail links)

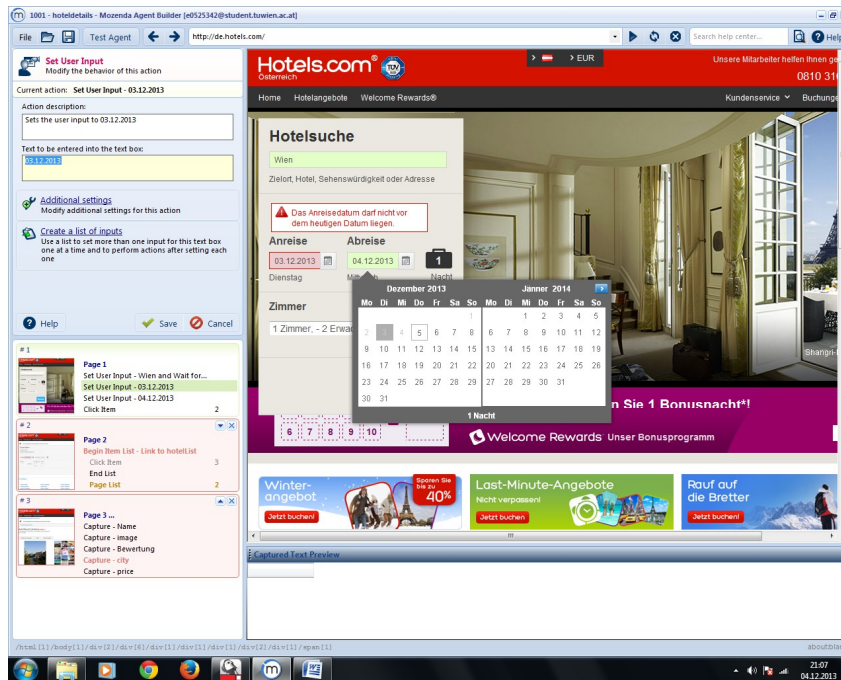


Fig 10: On the left upper site you can define user inputs for input variables

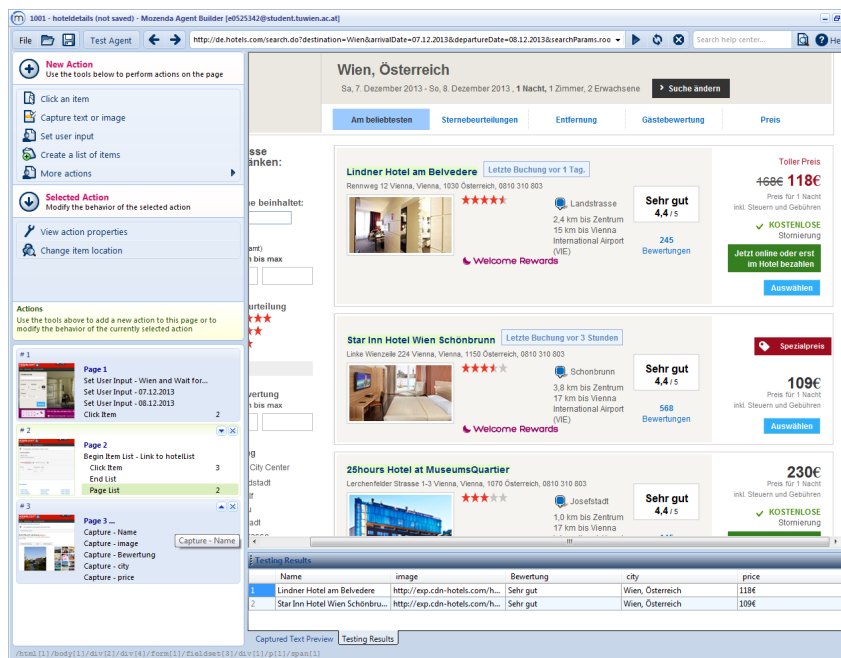


Fig 11: After building the agent, you can test it locally. Results are displayed at the bottom of the software.

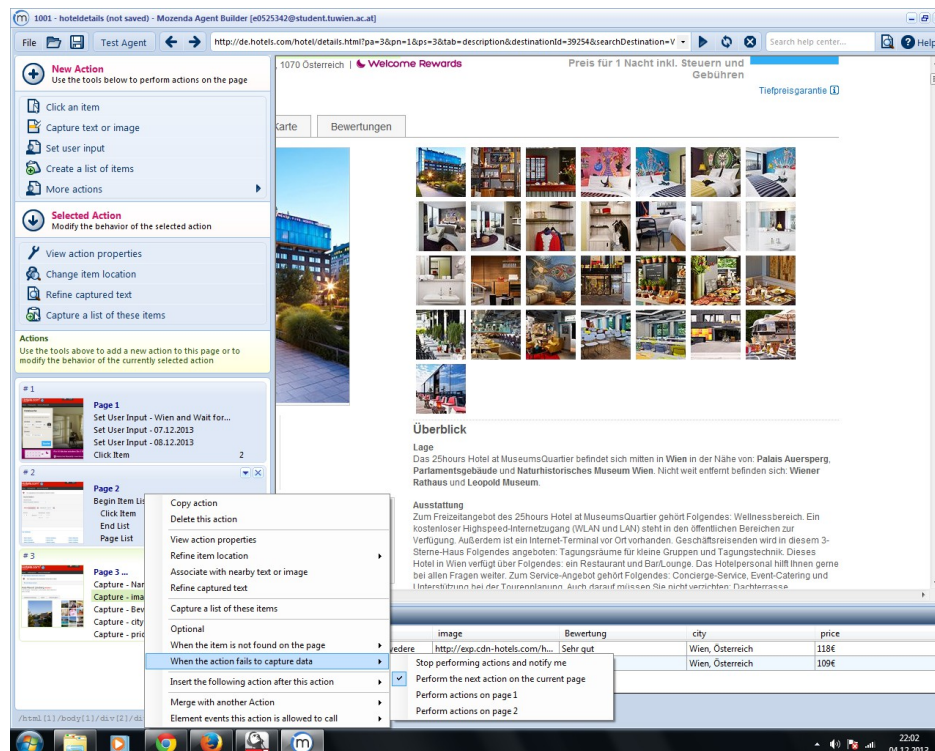


Fig 12. If condition: if action fails to capture an image from the detail link, the agent continues with performing the next action on the current page.

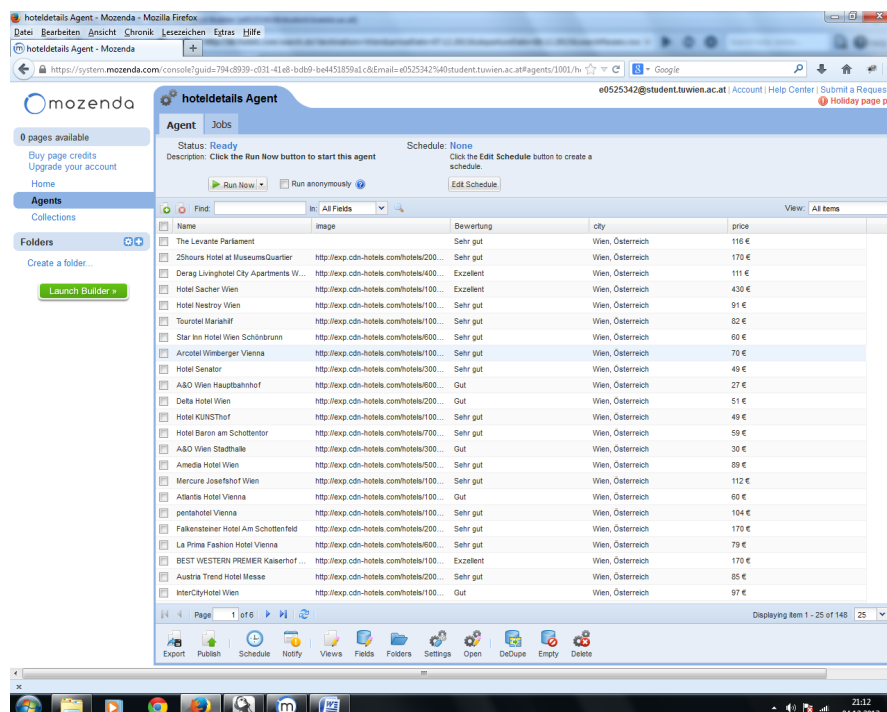


Fig 12: the Mozilla Web Console controls Agents (run, schedule) and Collections (filled by the agent)

Now we answer **additional questions**. In general, both wrappers were easy to build and quite intuitive to use. When using import.io, we had to extract detail information by using a second extractor and querying the data using the python script. On the other hand, we couldn't query mozenda data using the REST API directly, because this is not imported for trial accounts, so we had to do the extraction and export manually.

*** Document Model (DOM, Text, anything in addition to these)**

Import.io: The DOM is hidden from the user. One cannot see the DOM of the webpage. The only option to capture data is to use the mouse.

Mozenda: You can activate the feature, that DOM is visible in a control window at the bottom of the agent. If you click on an object, the specific DOM-entry is highlighted. But i didn't need to know about the DOM tree in this exercise.

*** Form Filling and Macro Recording**

Import.io: Quite natural I think. You navigate to the connector and can then start to record the form filling. Having seen the Lixto Visual Developer this is very restricted and there is not much to configure. The only interaction you can parameterize later are text fields.

Mozenda: Very easy and intuitive, took about 15 minutes to set up the whole wrapper.

*** Transform original tree structure**

Import.io: Elements can be hidden to more easily capture elements that are overdrawn. It occurred to me that this feature does not work in all circumstances.

Mozenda: feature not needed, most of the DOM functionality is hidden from the user.

*** Natural Language Processing Support**

Import.io: Limited. It is easy to learn certain parts of a text or phrase if it has an easy to find structure. It was quite hard to extract whole paragraphs of descriptions in the hotel details, and it did not really learn the description in the end (5 examples+).

Mozenda: Using regular expressions is supported

*** Heuristics and Rule Generation**

Import.io: Sometimes the recognition of some rules does not work very well with only a few examples. It sometimes takes up to 5 examples until the recognition of the attribute is learned.

But as far as I have worked with import.io it is very robust after it is correct.

Mozenda: Recognition of rules is very simple and robust. Works after choosing two examples. If a rule cannot be applied (because website changes for example), a warning is displayed.

*** Image Recognition**

Import.io: Images are handled quite nice. As the page is rendered directly one can easily extract the images. Before one can do that the data attribute has to be set to the type „image“.

Mozenda: also very nice, images are displayed directly when selected as output (Testing results), they can also be downloaded.

*** Scripting**

Import.io: While the rule learning is in progress there is not many parameters you can tweak.

But when the rules are completed you can access a very rich API in alot of different languages to aquire the data. It should be mentioned that the tool itself to genereate the wrapper for the cloud is really limited. That is why the extraction of the detail page has to be done with a seperate extraction call to the API.

Mozenda: no chance to learn the API calls since it cannot be used by a free account. Seems to be very powerful, agents can be steered and collections can be queried using constructed links.

*** Ajax Support and DOM Freezing, DOM Event Support**

Import.io: At first it always tries not to use the javascript engine. There are alot of dialogs asking you "Does the site look as expected?". If not it is reloaded with javascript enabled. This I think is quite good idea in terms of resources. But this fact makes extraction quite a tedious task when the javascript has to be enabled.

Mozenda: Agent waits for ajax-requests after loading content. Seems as if just a fixed time is waited, no matter how long the ajax- call needs to load data.

*** Input and Output Formats**

Import.io: Rich set of outputs. The wrapped data can be downloaded as EXCEL,HTML,JSON or CSV and by exposing a REST API those can be processed by any programming language.

Mozenda: input parameters: text; output format: CSV, TSV and XML

*** Control Browser Settings like User Agent**

Import.io: Cannot be changed. There are not many settings... At all.

Mozenda: Agents can be started, paused, restarted; time schedules can be defined

*** Parameterization**

Import.io: Can be specified when the is fetched from the REST API. This is quite conveyient but only supports textfields, not radio buttons, checkboxes, ...

Mozenda: can be specified when using the API (not working for free account)

*** Iteration, Conditions, Loops**

Import.io: Can only be done using a script backend.

Mozenda: all features included but hidden from the user (like pagination, if conditions)

*** Robustness and Adaptation**

Import.io: Not that robust, many times data is not extracted because the rules learned often could not be applied. Adaptation is really a problem. If you later change/add one attribute you have to train every attribute again in every example. This is not practicable in practice.

Mozenda: very robuts, easy to adapt

*** Automated Steps, Machine Learning (from multiple examples)**

Import.io: As mentioned earlier this is not practicable in practice because it certainly fails on hard tasks if one example is changed all the others have to be relearned.

Mozenda: very intuitive, little steps and examples needed

*** Storing screenshots / html source to file system**

Mozenda: capturing a screenshot is possible, but i couldn't find html source to file system functionality

Import.io: I don't think this is implemented in import.io.

*** Performance and Scalability**

Import.io: Decent. Compared to open dapper it is a little faster.

Mozenda: quite good, but WebL was faster

*** Ease of Use**

Import.io: Certainly not a tool I would use again for this course and neither in a real world example. You can do simple things very easily but it is tedious (when examples have to be relearned) and soon is impractical.

Mozenda: very easy and intuitive

*** Proxy Variations**

Import.io: No way to configure a proxy in this tool.

Mozenda: couldn't find this feature

*** Captcha Support**

Import.io: I don't think there is captcha support.

Mozenda: i couldn't find this functionality

*** Execution Environment**

Import.io: There is the GUI tool which is used to learn the rules and later deploy them as individual 'apps' in their cloud computing environment. There the data can be queried using any language you like. The latter I found very nice to use.

Mozenda: agent builder is the local software for testing the agent, but to extract data the agent is executed on the mozenda server