Hasso Plattner Institut
IT Systems Engineering
Universität Potsdam

Captcha Service

Documentation

Potsdam, February 2017

Supervisor

Prof. Dr. Christoph Meinel,
Christian Bartz

Internet-Technologies and Systems Group

Abstract

The increasing numbers of bots, especially crawlers, within the World Wide Web has been a major concern for several years now. Over the years, different approaches to tackle bots were implemented and tested. One of the major solutions for dealing with bots in the recent years were Captchas. Originally, giving users specific tasks to solve, which bots would be unable to solve, was the main idea. Through distortion and other obstacles, Captchas were improved against algorithmic solutions.

The potential of million online users solving Captchas was quickly realized. Difficulties in identifying words or objects in images using computers, could be solved using the combined solutions of Captcha users. We implemented our own Captcha Service in order to allow researchers and scientists to get their own datasets labeled using on line users.

Contents

1. Introduction

1.1. Motivation

Researchers and scientists are lacking the time and capacities to label their data, which is often further needed in order to advance other technologies. Using required authentication processes for online users, we are able to utilize huge amounts of free labor. Our main goal was building a straightforward service for researchers and scientists to allow precise data labeling.

Therefore, a simple integration for web services was also wanted.

2. Related Work

In preparation of creating an own Captcha service we searched the Internet for existing ideas and implementations of systems used for data labeling and machine learning purposes. The first popular approach was the Soylent Grid paper which was published in 2007 [1]. Although there were never any popular implementations of the ideas, the paper provided several different approaches for data labeling. They were mostly based around the idea of object recognition in images, e.g. by clicking on objects or drawing rectangles around it. Other proposals were directed to object recognition, where users had to name objects displayed in certain images.
Another paper which was published just a year later, dealt with text recognition and described the concept which was implemented in reCAPTCHA v1 [2]. The system was created by Google in order to solve problems in the "Google Book Project). Two different optical character recognition (OCR) algorithms were used to translate images of scanned pages into digital texts. The solving of Captchas was used to identify words which could not be deciphered clearly by the OCR algorithms. Because of its detailed documentation and its successful usage this method was ideal to be implemented and therefore the first Captcha type which is supported by our system.

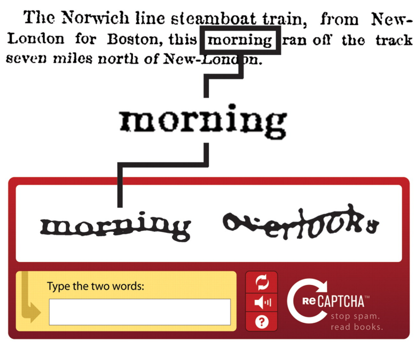
The Norwich line steamboat train, from New-London for Boston, this morning ran off the track seven miles north of New-London.
morning
morning overlooks
Type the two words:
reCAPTCHA
stop spam. read books.

Figure 1: The reCAPTCHA approach explained in one picture

[1]http://vision.ucsd.edu/sites/default/files/icv2007.pdf
[2]http://science.sciencemag.org/content/321/5895/1465.full

3. Architecture

The CaptchaService uses a Model-View-Controller to distinguish between data and it's representation. The views are included in the `views.py`-file and process requests made by the client, the third party web app and the web interface. The data model is represented in the `models.py`-file and manages *CaptchaTokens*, *CaptchaSessions* and the connection to the database. The System is designed for simple expandability and uses inheritance to simplify the introduction of new captcha types.

3.1. Models (models.py)

The models consist of two main classes, the *CaptchaToken* and *CaptchaSession*. An overview is given in the class diagram in figure 2.

The class *CaptchaToken* represents a single image, that is part for a captcha, e.g. a single word, that needs to be written down by the user in order to solve the captcha. The class *CaptchaSession* represents a complete captcha challenge a user has to solve, e.g. writing down the words shown on all images. Each type of captcha challenge provided by the service is represented by a subclass of *CaptchaSession* and *CaptchaToken*. Currently two kinds of Captchas, Image-Captchas and TextCaptchas, are supported.

All that needs to be done for implementing a new type of captcha challenge is to create a new subclass for *CaptchaToken* and *CaptchaSession* and implement specific functionality in these subclasses. Which methods and attributes need to be added in the new subclasses is listed in the "Attributes and Methods implemented in the subclass"-paragraph.

All instances of a *CaptchaToken* or *CaptchaSession* are saved in the `db.sqlite3`-Database.

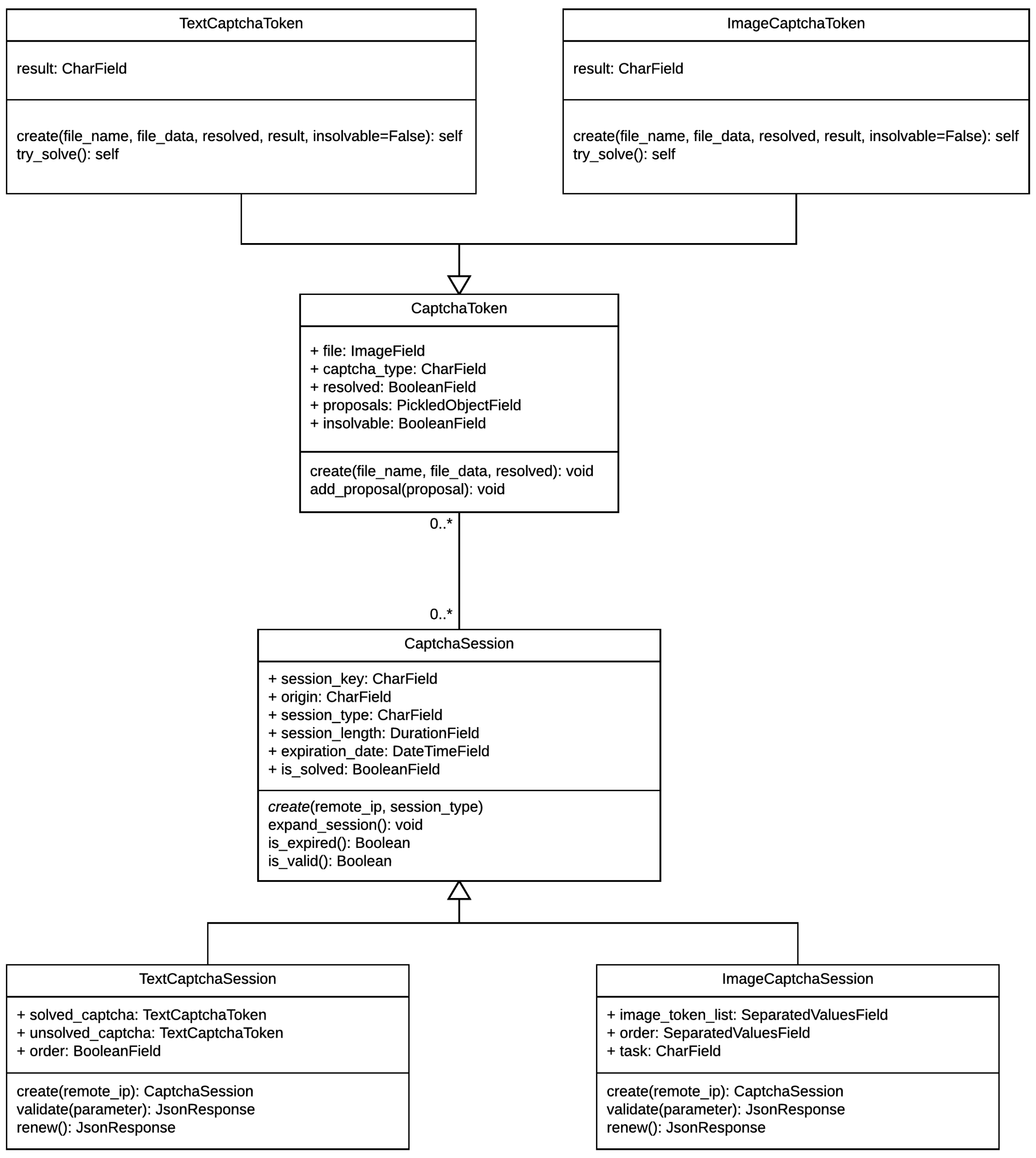
TextCaptchaToken
result: CharField
create(file_name, file_data, resolved, result, insolvable=False): self
try_solve(): self
ImageCaptchaToken
result: CharField
create(file_name, file_data, resolved, result, insolvable=False): self
try_solve(): self
CaptchaToken
+ file: ImageField
+ captcha_type: CharField
+ resolved: BooleanField
+ proposals: PickledObjectField
+ insolvable: BooleanField
create(file_name, file_data, resolved): void
add_proposal(proposal): void
0..*
0..*
CaptchaSession
+ session_key: CharField
+ origin: CharField
+ session_type: CharField
+ session_length: DurationField
+ expiration_date: DateTimeField
+ is_solved: BooleanField
create(remote_ip, session_type)
expand_session(): void
is_expired(): Boolean
is_valid(): Boolean
TextCaptchaSession
+ solved_captcha: TextCaptchaToken
+ unsolved_captcha: TextCaptchaToken
+ order: BooleanField
create(remote_ip): CaptchaSession
validate(parameter): JsonResponse
renew(): JsonResponse
ImageCaptchaSession
+ image_token_list: SeparatedValuesField
+ order: SeparatedValuesField
+ task: CharField
create(remote_ip): CaptchaSession
validate(parameter): JsonResponse
renew(): JsonResponse

Figure 2: Class diagram representing the classes used for the generation of captchas. The two main classes *CaptchaToken* and *CaptchaSession* are shown in the center. All other classes inherit from one of the superclasses.

3.1.1. CaptchaToken

The class *CaptchaToken* are the basic units of the captcha service. Data, that shall be labeled by the captcha service is saved as an *CaptchaToken*. Multiple *CaptchaTokens* are combined to a captcha challenge, when a new *CaptchaSession* is created.

Attributes and Methods implemented in the superclass

Attributes:

- `file`: Image, that is represented by the CaptchaToken.
- `captcha_type`: String, that defines the type of captcha the token can be used for. Currently "text" for Textcaptchas and "image" for Imagecaptchas are supported.
- `resolved`: Boolean, that indicates, if the solution for a *CaptchaToken* is known or not. A `0` means the token is unsolved and a `1` means the Token is solved.
- `proposals`: Dictionary, that stores the possible solutions suggested by users of the captcha service and how often each solution was suggested.
- `insolvable`: Boolean which indicates, that a token is not solvable by clients of the captcha service. This value is set to `True`, if there are many proposals for a *CaptchaToken* without one proposal having the majority of the votes. For more information see section 6 (solving algorithm).

Methods:

- `create( file_name, file_data, resolved)`: Responsible for basic configuration, that need to be done for all kinds of tokens, when they are created. Only used for super-calls in the `create()`-method of subclasses.
- `add_proposals(proposal)`: Adds a new suggested solution to the `proposals`-

dictionary, or increments the counter for an already suggested proposal.

Attributes and Methods implemented in the subclass

Attributes:

- `result`: Saves the correct solution for a token. Data type differs between different subclasses, e.g. *TextCaptchaToken* saves a string and *ImageCaptchaToken* saves a boolean.

Methods:

- `create(file_name, file_data, resolved, result, insolvable=False)`: Responsible for configuration of all attributes of the *CaptchaToken*. Returns a *CaptchaToken*.
- `try_solve`: Responsible for finding the correct solution for a *CaptchaToken* based on the values saved in the `proposals`-attribute.

3.1.2. CaptchaSession

Represents an instance of a captcha challenge, that needs to be solved by a certain client. A *CaptchaSession* consists of multiple *CaptchaTokens*, that are chosen randomly in order to create different challenges dynamically. Each Sessions corresponds to one of the supported types of *CaptchaTokens*. The tokens chosen are a mix of solved and unsolved tokens, in order to make it possible to validate the session and label new data.

Attributes and Methods implemented in the superclass

Attributes:

- `session_key`: String, that serves as primary key to identify each session.

- `origin`: String, that holds the IP address that requested the captcha challenge. It is used match requests made by the client to the corresponding session.
- `session_type`: String, that defines the kind of captcha challenge, the client has to solve. Currently "text" for Textcaptchas and "image" for Imagecaptchas are supported.
- `session_length`: Timedelta, that stores the amount of time, a session can be validated.
- `expiration_date`: Datetime, that stores the date, after which a *CaptchaSession* can not be validated anymore.
- `is_solved`: Boolean, that stores, if a *CaptchaSession* was already solved by the client.

Methods:

- `create(remote_ip, session_type)`: Responsible for basic creation of a *CaptchaSession* of the requested type for the given IP address. Only used for super-calls in the `create()`-method of subclasses.
- `expand_session()`: Responsible for changing the `expiration_date` of a *CaptchaSession* so that the session is valid for another `session_length`.
- `is_expired()`: Responsible for checking, if a session is expired. Returns `True` when the current date is before the `expiration_date`.
- `is_valid()`: Responsible for checking, if a *CaptchaSession* was successfully validated by a client and is not expired. In that case `True` is returned.

Attributes and Methods implemented in the subclass

Attributes:

Each session needs to store the tokens, that were used for creating the session and additional information, that is needed for validating the answer given by the client. This can differ for every captcha type.

TextCaptchaSession:

- `solved_captcha_token`: *TextCaptchaToken*, that is already solved and is used as a control word for the session.
- `unsolved_captcha_token`: *TextCaptchaToken*, that is not solved and shall be identified by the client.
- `order`: Boolean indicating the order, in which the two tokens are displayed to the client.(0 -> solved, unsolved 1 -> unsolved, solved) It is needed to map the answers given by the client to the right tokens.

ImageCaptchaSession:

- `image_token_list`: List of *ImageCaptchaTokens*, where all tokens used for the session are saved.
- `order`: List of Booleans, that indicates which token in the `image_token_list` is solved. (0 -> unsolved, 1-> solved)
- `task`: String, that saves the object that should be detected in the captcha challenge, e.g. cat, if cats should be identified.

Methods:

- `create(remote_ip)`: Responsible for creating a *CaptchaSession* and returning the created session to the corresponding `view`, and a JsonResponse with the parameters needed to render the captcha challenge in the front end (e.g. urls of pictures that are shown in the challenge). Depending on the type of the *CaptchaSession* the function chooses a mix of multiple *CaptchaTokens* with some of them being unsolved and some of them being solved. The unsolved *CaptchaTokens* are data that shall be labeled and the solved *CaptchaTokens* are used for checking, if the answer for the *CaptchaSession* is correct.
- `validate(parameters)`: Responsible for validating the solution for a CaptchaSession and returning the created session to the corresponding

view. The solution suggested by the client is included in the parameters. The method checks, if the suggested solutions matches the solution for the solved *CaptchaTokens*. If the solution given by the client is correct, proposals are added for the unsolved *CaptchaTokens* and try_solve() is called on these tokens. If the solution is false new *CaptchaTokens* are chosen for the *CaptchaSession* to prevent brute forcing. Returns a JsonResponse with information whether the session is valid or not and a list of image-URLs that shall be rendered in the session.

- renew(): Responsible for exchanging the *CaptchaTokens* of a *CaptchaSession*, to create a new challenge or the same session. Returns a JsonResponse of the updated information needed to render the session in the front end.

Session length and expiration

A *CaptchaSession* expires after a certain amount of time in order prevent replay attacks. For this reason is_valid() always returns false after the expiration date is over. The *CaptchaSession* however remains in the database indefintely, because Django does not provide an elegant method to automatically delete session after expiry. Therefore the CaptchaService provides a Django command, which is called delete_timeouted_sessions and on execution removes all timed out *CaptchaSessions* in the Database. This can be paired with a cron job so that timeouted sessions will be in short intervals.

3.2. Views (views.py)

The views handle POST- and GET-Requests made by the Client, third party web application and the web interface.

List of Requests handled by views:

- request(request): GET-Request called by the client, when a new session is requested. Chooses a type for the session randomly, calls create-Method for the session implemented in models.py and directs the response of session.create() to the client.

- `validate(request)`: POST-Request called by the client, when a solution for a captcha challenge is submitted. Retrieves the the corresponding *CaptchaSession* from the database, calls the validate-Method of the session and directs the response to the client.
- `renew(request)`: POST-Request called by the client, when a new challenge for an existing *CaptchaSession* shall be provided. Retrieves the the corresponding *CaptchaSession* from the database, calls the renew-Method of the session and directs the response to the client.
- `upload(request)`: POST-Request called by the web interface, when new files are uploaded to the captcha service. Extracts the files from the zip-file and creates tokens corresponding to the provided data.
- `download(request)`: GET-Request called by the web interface, to retrieve tokens and solutions from the database. Collects *CaptchaTokens*, that meet the requirements specified in the web interface, compresses them to a zip-file and returns the file.
- `getTask(request)`: GET-Request called by the web interface, to get all possible tasks for an *ImageCaptchaSession*. Gets all tasks from the database and returns them as a list in a JsonResponse.
- `validate_solved_session(request)`: GET-Request called by the third party web application, to check is a *CaptchaSession* was successfully validated by a client. This prevents clients from being able to bypass the captcha challenge.

4. Image Distortion

The fact that images for text Captchas are provided by users makes it impossible to tell if those images are easy to recognize for bots and are therefore safe to be used as Captcha token. In order to complicate the recognition of the Captcha token the systems uses an image distortion algorithm which is automatically applied to all uploaded text Captcha tokens.
The image distortion algorithm consists of two steps: the drawing of a horizontal line and a wave transformation.
In the first step it places a horizontal line in the middle of the image which is colored with the dominant color of the whole picture. Afterwards this line will be transformed together with the rest of the image.
The frequency as well as the amplitude of the wave which will be applied to text are dependent to the height of the image. Furthermore the frequency depends on the width of the image so that one wavelength is at least as wide as the image itself. In addition to this both, the frequency and the amplitude, are modified by a random value in order make every transformation unique.
Everything that is shifted out of bounds will be cut off. Additionally the pixels which were located at the bottom and the top of the the original picture will be stretched out vertically to fill the space which was emptied due to the transformation.

5. Solving Algorithm

In order to provide a value for the researchers which add data to the Captcha service, the system has to label the uploaded images. This becomes possible due to the solving algorithm, which determines the label based on the given user inputs.
In case of text Captchas the algorithm needs at least three users which solved the Captcha correctly. If three or more suggestions match, the image is marked as solved and labeled accordingly. However the token is identified as unsolvable if there are six or more proposals but no more than two of them match. This ap-

proach relatively similar to the concept reCAPTCHA uses. In a paper [3] that was published it was stated, that in most cases three human resolutions are enough to label the image reliably.

The method for labeling image Captchas is similar to the one used for texts. The main difference is the fact that the proposals for these are limited to *true* and *false*, are they suiting the specified task or not. Therefore the algorithm checks if at least four resolutions match and also declares a token as unsolvable if more the six suggestion are given but failed to produces four that match. It was decided to raise the bar for labeling a picture from three to four, because it is more likely to falsely select an image due to a wrong click.

[3]http://science.sciencemag.org/content/321/5895/1465.full

6. Evaluation

As proven by reCAPTCHA the solving algorithm is accurate most of the time. However there remains a small chance that images are labeled incorrectly. Certainly there is no way to completely eliminate the possibility of inaccurate image labeling. It would be possible to minimise this probability by further increasing the number of needed proposals, but this would slow the labeling process down and therefore result in fewer labeled images over time. The current implementation provides an acceptable trade-off in order to offer satisfying results for researchers, which rely on this service for data labeling.

Another major feature of the CaptchaService is the ability to reliably combat bots and spammers. The security features, which are in place right now will be enough hold off most of the attacks. However one could argue, that it is not sufficient to renew the Captcha after a wrong proposal and validate the solved session of a user. Further security mechanism, such as blocking of ip addresses after consecutive inaccurate solutions could further increase the security of the system. Nevertheless it will also worsen the user experience of clients that fail to solve Captchas reliably. This could ultimately lead to users not visiting the site again. Therefore the current implementation does not rely on any further security mechanisms and thus offers an optimal trade-off between user experience an security.

7. Future Work

With the thought in mind of building an easy expandable service, the logic consequence would be on focusing on different Captcha types. In the process, key factors such as access for disabled users can be tackled, e.g. by implementing audio Captchas. The accessibility for all users is important to allow for an unrestricted usage of the web services. Another aspect would be expanding the web interface. The option of downloading solved and unsolved Captchas can be specialized by selecting specific upload times or certain time spans. A feedback of the labeling progress within a task would also be another great tool.

A. Appendix

Appendix