Milestone Report 2

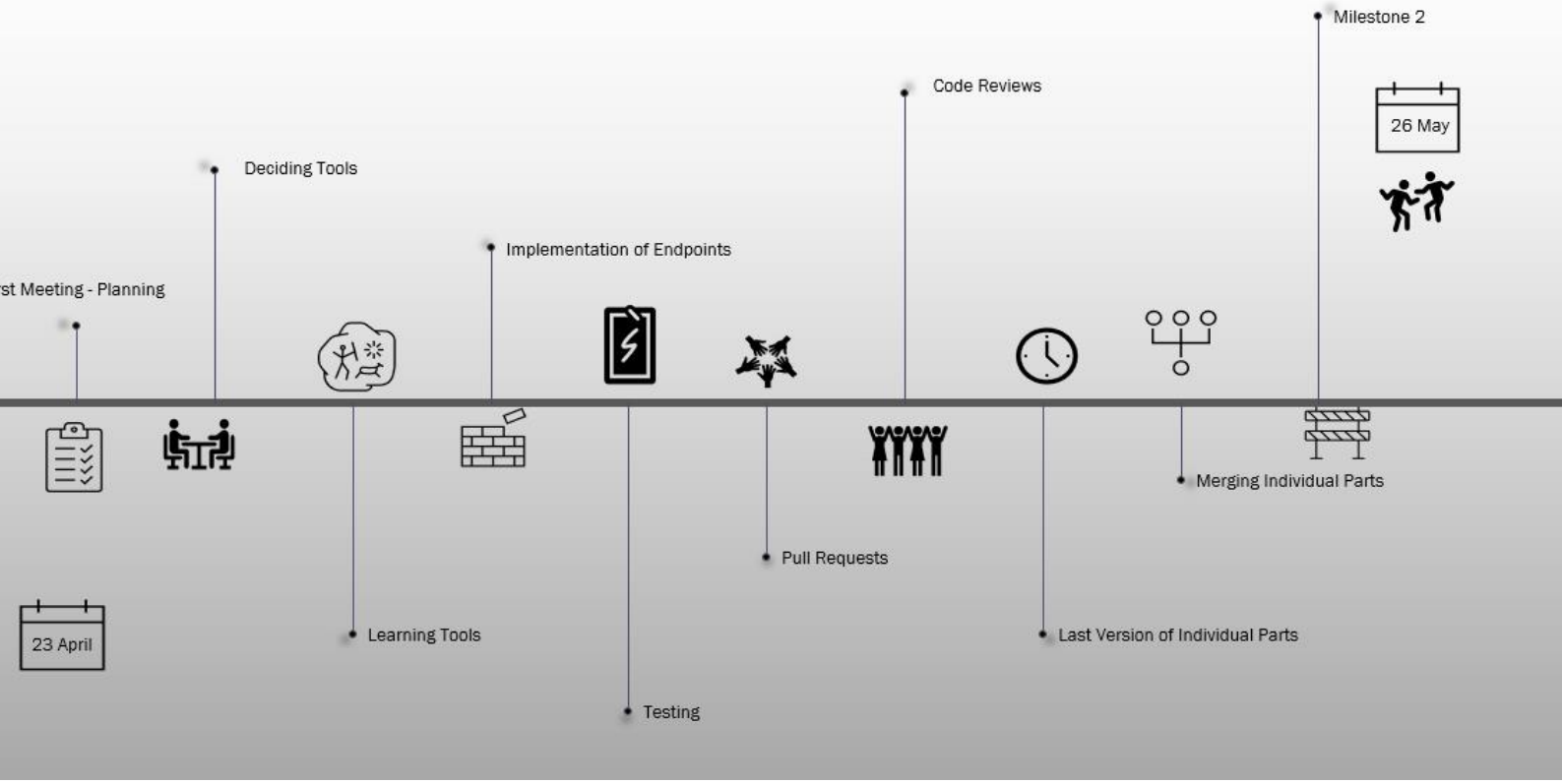
26.05.2020

Platon – The Academic Collaboration Platform

Hasan Ramazan Yurt, Burak Omur, Ahmet Dadak, Kerem Uslular, Halil Umut Ozdemir

Alperen Divriklioglu, Ertugrul Bulbul, Meltem Arslan, Mehmet Temizel, Oyku Yilmaz

Group 7



Contents

Executive Summary

Introduction – What is this report?

In this project we are expected to build a RESTful API to prepare us for the implementation of our academic collaboration platform project – PLATON. We were free to identify project details such as frameworks and external APIs that we will use. In this milestone report, we will explain the roadmap of our project and the details of the practice-app.

Work done so far

Before the project:

1- It was learned what RESTful API is, how to use GIT commands, collectively contributing a project using GIT.
2- Informed about frameworks and IDE's.
3- It was decided to use Django as a framework, MySQL as a database management system, Poemist, Datamuse, nytimes, dataAtWork, NewsAPI and Jokes One as external API's
4- Decided tools were installed and usages of the tools were learned.

During the project:

5- Endpoints were implemented by each team member in Django.
6- Endpoints were tested by each team member.
7- Code documentations and API documentations were prepared for each endpoint.
8- Pull requests were created and reviewers were added to the requests. The codes were reviewed by the assignees. Feedbacks were given by them and the branches were finalized by making necessary arrangements.
9- The branches were merged and bugs that occurred at this stage have been fixed.
10- Docker image was created.
11- A database was generated from AWS. Our application was deployed to AWS.

After the project:

12- Milestone 2 Report was prepared.

Road ahead

The result we obtained from practice-app met our expectations, but thanks to this project, we decided to re-evaluate the tools we selected. This project provided us as an introductory project experience as it has similar features to our original project, and it provided an opportunity to see our shortcomings in the planning phase of our original project. Project plan will be updated in line with the experience we gained from practice-app.

Challenges

At the beginning of our assignment, the selection process did not proceed very smoothly, as we did not have enough knowledge to compare tools. Lack of experience caused us to spend more time than it should be while solving the problems that occurred during the installation of the tools we chose. Lack of face-to-face communication slowed the advancement of the project. Most of us had a painful experience, especially when using Django, so we decided not to continue with this framework next semester. As individual works differ in using the environment, we spent a lot of time fixing conflicts at the merge stage.

List and Status of Deliverables

All deliverables listed below are complete. Completion date implies the date of the last change made on them.

DELIVERABLE	COMPLETION DATE
Code Documentation	25 May 2020
API Documentation	25 May 2020
Practice-app	26 May 2020

Evaluation of Deliverables

Code Documentation

Code documentation is one of the deliverables that make our work easier when working together. It was descriptive for the parts we do not understand when doing code review. It will help us remember some parts of this project more easily while doing our next term project - Platon. Also, it will help the non-members of the project while reviewing it.

API Documentation

API documentation is a deliverable containing information about the functions, classes, return types, arguments etc. It helped the coders of the front-end part of the project in understanding their collaboratives' codes. The documentation also helps us in the next semester to interact with our previous project more easily. In the last part of Milestone Report 2, the API documentation will be described in detail.

Practice-app

There were two main tasks for practice-app: Front-end development and back-end development. First, we developed the back end by dividing it into ten parts and implemented them individually, so each team member had an endpoint. After testing our individual codes, we requested code review, and created pull requests. We have finalized the codes according to the feedback provided. Each team member wrote their API and code documentations. In the merge stage, we had some difficulties due to the conflicts between individual codes, so we had to handle the conflicts while merging. After merging individual parts, the back-end part of practice-app was tested and then we continued with the implementation of the front-end. Ahmet and Ramazan developed the front-end part. Before deploying our application, we tested all the endpoints including frontend using unit tests and Postman. In the end, Burak created a docker image and sent it to DockerHub, and Umut deployed the application to AWS.

Evaluation of Tools and Processes

GitHub

Github is a platform that meets many requirements of the project creation and development process for collective studies. Thanks to the Git feature, it enables efficient and up-to-date interaction between local - online environments, also between the collaboratives. Github allowed us to systematically follow the changes made in the codes and the reviews made on the pull requests. Thus, we were able to find out where the error was more easily by looking at the previous versions for the problems we encountered after testing the code. The ability to write our codes in different branches allowed us to work independently. But until the merge stage, we could not see the conflicts between our codes unless we constantly followed each other's codes. So, we had to solve this problem while doing the merge.

Django

We used Django as a framework. It is a free and open source web application framework written in Python. Django uses the MVP (Model View Template) architecture. The model is the layer where the database operations are performed. We model our database related operations in this layer and use this layer when it is available. SQL commands are not needed to create a database in Django. A database is created in a language specific to Django and Django is used with its own database comment. View is our development part. All our python codes are located here. It is a bridge between other layers, and we draw the necessary part with our Python codes and use it in this layer. Template is our design and presentation layer. The web page or elsewhere is concerned with how our page will look.

MySQL

MySQL is a relational database management system based on SQL – Structured Query Language. It is an open source software supported by Oracle. You can store data in tables, create indexes on the data and query the data using SQL queries.These are the fundamental functions of any relational database management system .RDBMS and MySQL are generally thought to be the same due to the popularity of MySQL but MySQL is one of many RDBMS software options. It allows you to manage

relational databases.

Visual Studio Code

Visual Studio Code provided us a good experience as a source code editor especially thanks to its useful extensions. Selecting the folder where we will work and seeing the files from the navigator with the hierarchy saved us from getting lost among the files. Thanks to the Live Share extension, we were able to work on the same file at the same time. When we needed to work on two different files at the same time, we could do side-by-side editing.

Postman

We used Postman a lot when developing our endpoint. Postman is one of the easiest and most common tools used to test API. While testing the endpoints, it makes it very easy to get responses without using frontend by simulating requests. For these reasons, postman is a very functional tool for backend developers.

Sourcetree

Even if not all members of the group prefer, sourcetree and its interface were very helpful especially for those who are not used to using command line. We could see the files we made changes on the screen in order, so we didn't have to keep in mind which files we should push. By selecting the modified files, we could perform pull, push, command operations independently of the complexity of the command line by clicking their buttons on the same screen.

Docker

Docker works as a machine that runs python on it, it runs our django application. Also, we can reach our repository using Docker.Hub from anywhere and run it with one line code.

Zoom&Discord

Zoom and Discord are the applications that helped us the most because we did not have the opportunity to communicate face-to-face while developing this project. Sometimes, the need for a good internet connection for screen sharing disrupts our communication but thanks to this feature, we provided better communication while planning the project, explaining and commenting on our codes and solving the errors in the tests results.

Summary of Work Done Individually

TEAM MEMBER	CONTRIBUTION
Burak Ömür	• I have implemented POST and GET methods for registration api in "rest_api/register/register.py" file with endpoint api/register and also frontend with api/register/fe endpoint. • There are 4 functions in api as getPoem() to get random poem from external api, getJobName() to get normalized job name from external api, isValid() to check whether given input is valid or not, and register_api() to control api. Also, there is a function named register_page() that renders a html to show a frontend for api endpoint. • I have prepared 8 unit tests to test valid and invalid inputs to api, also one to test get requests. Unit test can be found in rest_api/register/register/test.py • I have also reviewed 3 pull requests that are “practice-app-logout”, “practice-app-user-delete”, and “practice-app-user-update”, then gave feedback to them by clearly pointing out the necessary parts in code. • I have helped to merge branches that have conflicts with practice-app-development branch. • I have prepared documentation of my code using pydoc and documentation of my endpoint, both html and txt format can be found in the rest_api/register/register directory. • I have created Docker Image of our application to deploy.
Meltem Arslan	• I have implemented POST method in "rest_api/update_user/update_user.py" file with endpoint api/userUpdate • I implemented the methods in the file “rest_api/update_user/update_user.py”. The “updateUser” function updates the database with given input, “isValid” function checks the validity of that input and “getJobName” function returns the uuid of the job which’s name was given, using an external api. • I have prepared a unit test to test the validity of given inputs’ format and given password to match the password with a registered user. Unit test can be found in rest_api/update_user/test/test.py • I have reviewed 3 pull requests that are “practice-app forgot password”(https://github.com/bounswe/bounswe2020group7/pull/77) , “practice-app register”(https://github.com/bounswe/bounswe2020group7/pull/67) and “practice-app joke”(https://github.com/bounswe/bounswe2020group7/pull/73), then gave feedback to them. One of my feedback was checking the matching of the passwords but not entering the null value(to Burak). The second one was the deletion of the pycache file which can cause conflicts at the merge stage(to Ertuğrul) and the third one was about the deletion of unnecessary imports such as re,copy, random, json, requests, render, Response (in forgot_password.py) • I documented my code and my API. The API documentation is in the following link under the update user heading :https://github.com/bounswe/bounswe2020group7/wiki/API-Endpoint-Documentation . The code documentation is in the following link under the documentation file, in the markdown format. https://github.com/bounswe/bounswe2020group7/tree/practice-app-user-update/practice-app/platon_api/rest_api/update_user . • I created Milestone Report 2 with Ertuğrul.

Halil Umut Özdemir	● I implemented the search functionality in our practice-app. It has only one endpoint which takes only GET requests and the endpoint is "../api/search/". ● The code that I wrote can be found in the "rest_api/search_engine" folder. Also, I made some changes in "rest_api/views.py". ● I write a class called searchEngine which can be found in the "rest_api/search_engine/search_angine.py". In this class there are 7 functions. semantic_related_list() gets seantşcly related word from an external API, get_parameter_list() parse the parameters which is given in the URL, verify_token() checks whether the token is valid or not, sort_output() sorts the output, get_stopwords() gets stopwords of English from an external API, get_job_name() gets the job of a user with the use of an external API, and search() function is the main function that search a user in the database. It makes a simple semantic search. ● I writed 15 unit test for the search endpoint. Some test cases tests invalid input responses of the endpoint, some of the tests the usual semantic search, filters and sorting criterias are working or not and there is one test case that tests the semantic_related_list() function. These tests can be found in the "rest_api/search_engine/test" directory. ● I made 3 code reviews which are "practice-app-login", "practice-app-delete-user" and "practice-app-frontend". I gave some feedback about the code and documentation in this code review. ● I solved merge conflicts and merged other branches to the "practice-app-development" branch. During this Öykü helped me a lot. ● After each merge operation Öykü and I tested the current situation of the code using Postman and the unit tests. ● I prepared documentation of my code using pydoc and documentation of my endpoint, both html and txt format can be found in the rest_api/search_engine/documentation directory. ● I created an online MySQL database for our app and also I helped to deploy the docker image to AWS.
Hasan Ramazan Yurt	● I implemented a translation module in our practice-app. API has only one endpoint which takes GET requests and returns a translated text. The end point is "../api/translation/<str:token>" ● The source code of API can be found in the "rest_api/translation" folder. ● I also made some changes in "rest_api/view.py" to call my API. ● In the "rest_api/translation" there are 3 files: "translation.py", "test.py", and "translation_document.md". "translation.py" is the main source file which has translate() function that translates from English to Yoda. ● I wrote "test.py" that is the test file that has 1 setup and 3 test functions: test_validTranslation(), test_keyErrorTranslation(), and test_noTokenInDb(). ● I created "translation_document.md" that is the documentation of the translation API. ● I reviewed three pull requests that are called practice-app-registration, practice-app-news, and practice-app-forgot-password. I opened my pull request. ● I have also implemented half part of the frontend. In frontend we reviewed all codes and integrated these codes to view.py. Also, I wrote some components of the website. ● I helped to merge some branches to "practice-app-development".

Öykü Yilmaz	● I implemented an API which deletes the logged in users' accounts. The DeleteUser class can be found under the directory rest_api/delete_user_t/delete_user_f. ● I implemented all three functions in that class, one is bestsellers(), which uses an external API from New York Times and returns a book recommendation. verifyTokenAnMail() function is also written by me and checks if the token and mail of the user belongs to a record in the user database. deleteUser2 function executes the SQL statement to delete that user and returns a book recommendation if everything goes right, and an error otherwise. ● I implemented 6 unit tests that are checking all the error messages and if token and mail is given properly, if the system deletes the user. They can be found under the directory rest_api/delete_user_t/unit_test.py. ● I updated the functions in the views.py to call the delete functions ● I reviewed three pull requests for the endpoints register, search and logout. ● I tried to help merge branches as much as I could. ● I documented my code and generated an html document with pydoc. That is in the same directory with my code - rest_api/delete_user_t ● After each merge operation I was a part of the group that tested the current situation of the code using Postman and the unit tests.
Ahmet Dadak	● I implemented an API endpoint api/news/ which allows users to get data from an external API called NewsAPI. ● I implemented a function called news_api() for this API endpoint in the file news.py in rest_api/news folder and it returns the recommended news by looking his/her field of study which is specified while registering. ● I implemented test_wrong_token() in test_news.py in rest_api/news/test folder. Since my API endpoint is valid for registered user, it need to check by looking token whether user logs in. This test is checking whether token is wrong ● I implemented the frontend part of the platon-api ● I made a code review of two pull request called practice-app-login and practice-app-update-user branches. I gave feedback by examining the code. ● I documented my code and generated a markdown document with pydoc. You can find the document under the directory "rest_api/news/documentation". ● I documented the API endpoint I implemented. It can be found at our GitHub repository under "API-Endpoint-Documentation".
Kerem Uslular	● I implemented an API which the user uses to login to the system. The login class can be found under the directory "rest_api/login/login.py". ● I implemented the login() function in the class Login, which takes an Http request as a parameter. This function then checks if there is a record with the given credentials in the user database. Finally if the user is valid, the function sets the corresponding row's token and redirects the user to the homepage. ● I prepared three unit tests that are checking a successful input and two invalid inputs. These are test_user_login(), test_failed_login() and test_failed login_with_missing_info(). You can find these unit tests under the directory "rest_api/login/tests/login_tests.py". ● I reviewed two pull requests for other endpoints. These are "practice-app-joke" and "practice-app-news". I gave feedback via comments accordingly. ● I documented my code and generated a markdown document with pydoc. You can find the document under the directory "rest_api/login/documentation". ● I documented the API endpoint I implemented. It can be found at our GitHub repository under "API-Endpoint-Documentation".

Mehmet Temizel	● I have implemented an API for forgot password under the directory "rest_api/forgot_password/forgot_password.py" . ● There is a function in the api as forgot_password to reset password if the user forget his password. However he must answer the secret question query correctly. ● I have prepared 6 unit tests to test valid input and invalid inputs(wrong mail, unmatchingPassword, answer_secret_question etc.) to the api. You can find the unit tests in rest_api/forgot_password/forgot_password/test.py ● I have also reviewed 2 pull requests that are “Practice app joke”, and “Practice app translation”, then gave some feedback to them about what wrong is and what nice is . ● I have prepared documentation of my code and documentation of my endpoint. You can find my endpoint documentation at API-Endpoint-Documentation page.
Alperen Divriklioglu	● I implemented the logout API endpoint of our app. ○ logout() method in rest_api/logout/logout.py. ● I prepared documentation for my implementation of logout endpoint. It can be found in rest_api/logout/logout_documentation.md. ● I prepared three unit tests for the logout endpoint. ○ test_logout_without_token ○ test_logged_in_user_logout ○ test_non_logged_in_user_logout ○ All the tests listed above are implemented in rest_api/logout/logout_tests.py as methods in LogoutTestCase class ● I reviewed the code for the “forgot password” endpoint (which was pushed to “practice-app-forgot-password” branch), I gave my feedback through the comments.
Ertugrul Bülbül	● I implemented an API which returns a joke by using an external API. The Joke class can be found under the directory rest_api/joke/joke. ● There are 3 functions in api such as getJoke() to get the joke of the day from external api, verifyToken() to check if the token given is valid or not, joke_api() to handle requests and return joke as JSON. ● I have prepared 3 unit tests to test valid, invalid and without token cases. Unit test can be found in rest_api/joke/joke/test_joke.py ● I have also reviewed 2 pull requests that are “practice-app-search” and “practice-app-translation”, then gave feedback based on the errors I could find in their branch. ● I have prepared documentation of my code and documentation of my endpoint,it can be found in the rest_api/joke/joke directory. ● I created Milestone Report 2 with Meltem.

Deliverables

Code Documentations

Register Code Documentation

NAME rest_api.register.register - Created on MAY 16, 2020

DESCRIPTION

```
This script controls the registration api of PLATON_API, using django&mysql backend.
Endpoint description:
    http://localhost:8000/api/register/

    'GET':
        Returns a random poem using external api
    'POST':
        Gets dictionary from body.
        JSON Format : { 'name': "",                 string, Name parameter given by user
                        'surname': "",              string, Surname parameter given by user
                        'password1': "",            string, Password1 given by user
                        'password2':"",             string, Password2 given by user to check whether
Password1 is matched.
                        'e_mail':"",                string, Email parameter given by user
                        'about_me':"",              string, About me parameter given by user
                        'job_name':"",              int, job id parameter chosen by user
                        'forget_password_ans':"",   string, forget password answer parameter given by
user
                        'field_of_study':"" }       string, field of study parameter given by user

@author: Burak Omur, darktheorys
@company: Group7
```

FUNCTIONS

```
getJobName(name)
        where name is the job name taken from user while registration

        returns uuid of normalized job, or empty string

    This function takes a string and using an external api it normalizes and returns the uuid of that
job.

getPoem()
        return a python dictionary, json that includes random poem

    This function returns a random poet using an external api.

isValid(name, surname, password1, password2, email, about_me, job_name, forget_pw_ans, field_of_study)
        where 'name': string, Name parameter given by user
        where 'surname': string, Surname parameter given by user
        where 'password1': string, Password1 given by user
        where 'password2': string, Password2 given by user to check whether Password1 is matched.
        where 'email': string, Email parameter given by user
        where 'about_me': string, About me parameter given by user
        where 'job_id': string, job id parameter chosen by user
        where 'forget_pw_ans': string, forget password answer parameter given by user
        where 'field_of_study': string, field of study parameter given by user

        returns True if given values appropriate to insert in database, else False

    This function takes input parameters and checks them if they are valid and return the boolean
result.

register_api(request)
    where 'response': rest_framework response

    returns response from rest_framework

    This function only accepts POST requests, and if valid input is necessary inserts into database

register_page(request)
    where request HttpRequest to carry post and get requests

    This function takes a request and if POST, then it registers into system using another endpoint of
this api
    If get, it renders a form to register and also a random poem.
```

Search Code Documentation

NAME search_engine

DESCRIPTION

```
Created on MAY 17, 2020
This script controls the search functionality api of PLATON_API, using django&mysql backend.
Endpoint description:
    http://localhost:8000/api/search/

    'POST':
        Produces error
    'GET':
        Gets dictionary from body.
        JSON Format : {
                        token = "Your token will be here!!",
                        search_string = "The string that you want to search will be here",
                        filter = {"job" = "job_filter", "field_of_study" = "field_of_study_filter" },
                        sorting_criteria = "Sorting criteria will be here"
                        }
@author: Halil Umut Ozdemir
@company: Group7
```

CLASSES

```
builtins.object
    searchEngine

class searchEngine(builtins.object)
 |  In this class the Search Engine endpoint is implemented.
 |
 |  ...
 |
 |  Attributes
 |  ----------
 |  job_list : list
 |      a list of jobs that whose index is written in the database.
 |
 |  exact_match_score: int
 |      Semantic score of an exact match in search operation
 |
 |  num_of_semantically_related: int
 |      Number of words that search engine uses in semantice search for each token
 |
 |  sotring_criteria_list: list
 |      a list of sorting criteria that can be used in the search functionality
 |
 |  Static methods defined here:
 |
 |  get_job_name(job_uuid)
 |      where 'job_uuid': job uuid of a job
 |
 |      returns job name of given uuid
 |
 |      This function converts a job uuid to its name
 |
 |  get_parameter_list(request=None)
 |      where 'request': HTTP request that comes from the view class
 |
 |      returns a dictionary of the paramters if there is no problem about the paramters
 |
 |      This function is used to split the arguments of the endpoint in the correct order.
 |
 |  get_stopwords()
 |      returns list of stopwords + punctuation in English
 |
 |      This function returns a list of stopwords for English
 |
 |  search(request)
 |      where 'request' : HTTP Request that comes from the view class
 |
 |      returns a list of search results.
 |
 |      This is the function that searchs the user in our database
 |
 |  semantic_related_list(search_tokens, max_num_of_related)
```

```
|      where 'search_tokens': List of tokens that are given in a search string
|      where 'max_num_of_related': Number of words that will be chosen for a token
|
|      returns a list of tuples which contails (word,semantic relation point)
|
|
|  sort_output(output_list, sorting_type)
|      where 'output_list': the list of the results that come without sorting
|      where 'sorting_type': sorting type parameter
|
|      returns sorted version of the search result
|
|      This function sorts the output with a given sotring criteria
|
|  verify_token(token=None)
|      where 'token': string, 64 characther string that can be token
|
|      returns True if token exists in the DB
|
|      This function verifies the token if there exits.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  exact_match_score = 1000000
|
|  num_of_semantically_related = 5
|
|  sotring_criteria_list = ['name_increasing', 'name_decreasing', 'surnam...
```

Daily News Code Documentation

NAME

```
rest_api.news.news
```

DESCRIPTION

```
Created on MAY 15, 2020
Controls recommended news api of PLATON_API, using django&mysql backend.
Endpoint description:
    http://localhost:8000/api/news/<str:token>

    'POST':
        Produces error
    'GET':
        Gets dictionary from body.
        JSON Format : {
                        url = "url of the news",
                        imageUrl = "image's url of the news",
                        title = "title of the news"
                        }
@author: Ahmet Dadak
@company: Group7
```

FUNCTIONS

```
news_api(request, token)
    returns featured news article for the user's field of study

verify_token(token=None)
    where 'token': string, 64 characther string that can be token

    returns True if token exists in the DB

    This function verifies the token if there exits.
```

Login Documentation

NAME

```
login
```

DESCRIPTION

```
Created on MAY 23, 2020
This script controls the login functionality api of PLATON_API, using django&mysql backend.
Endpoint description:
    http://localhost:8000/api/login/

    'POST':
        Produces error
    'GET':
        Gets dictionary from body.
        JSON Format : {
                        e_mail = "Your email will be here.",
                        password = "Your password will be here, hashed for protection."
                        }
@author: Kerem Uslular
@company: Group7
```

CLASSES

```
builtins.object
    login

class login(builtins.object)

 |  In this class the Login System endpoint is implemented.
 |
 |  Static methods defined here:
 |
 |  login(request)
 |      where 'request' : HTTP Request that comes from the view class
 |      returns responses
 |      This is the function that verifies the user and gives the user a token.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

Logout Documentation

This method provides the logout functionality to the app.

Expected input: a GET request with "token" parameter in query string.

How it works:

1. It retrieves the token from the query string in the URL. (If the request was not sent with GET method or if there is no "token" parameter in the query string, a "400 Bad Request" error is returned.)
2. Whether there is a currently logged in user with the token in the request gets checked by a database query. If found, the token for the user in the database gets nullified. (If not, -namely, when there is no currently logged in user with the token- a "401 Unauthorized" error is returned.)
3. After the token for the user gets nullified, "You have succesfully logged out. Goodbye, see you soon!" message is returned with the 200 status code.

Forgot Password Documentation

NAME

```
rest_api.forgot_password.forgot_password
```

DESCRIPTION

```
Created on MAY 24, 2020
This script controls the forgot password endpoint of PLATON_API, using django&mysql backend.
Endpoint description:
    http://localhost:8000/api/forgotpassword>

    GET':
        Produces error
    'POST':
        Changes the registered user password.
        JSON Format : {
                        e_mail = "E-mail parameter given by user",
                        forgot_password_ans = "forget password answer parameter given by user",
                        }
@author: Mehmet Temizel
@company: Group7
```

FUNCTIONS

```
forgot_password(request)
    returns the response of whether password change succesfully or not
```

Update User Documentation

NAME update_user

DESCRIPTION

```
Created on MAY 27, 2020
This script controls the user information update functionality api of PLATON_API, using django&mysql
backend.
Endpoint description:
    http://localhost:8000/api/updateUser/

    'GET':
        Produces error
    'POST':
        Gets dictionary from body.
        JSON Format : { 'token': "",                string, token parameter given by the system when
user logged in
                        'name': "",                 string, Name parameter given by user
                        'surname': "",              string, Surname parameter given by user
                        'password1': "",            string, Password1 given by user
                        'e_mail':"",                string, Email parameter given by user
                        'about_me':"",              string, About me parameter given by user
                        'job_name':"",              int, job id parameter chosen by user
                        'forget_password_ans':"",   string, forget password answer parameter given by
user
                        'field_of_study':"" }       string, field of study parameter given by user

@author: Meltem Arslan
@company: Group7
```

FUNCTIONS

```
getJobName(name)
       where name is the job name taken from user while registration

    Returns uuid of normalized job, or empty string

    This function takes a string and using an external api it normalizes and returns the uuid of that
job.

updateUser(request)
       where 'request': HTTP request that is from the view class
    Accepts only "POST" requests and returns a response with status.
    This function updates the information of logged in user.

isValid(name, surname, password1, email, about_me, job_name, forget_pw_ans, field_of_study, user)
        where 'name': string, Name parameter given by user
        where 'surname': string, Surname parameter given by user
        where 'password1': string, Password1 given by user
        where 'email': string, Email parameter given by user
        where 'about_me': string, About me parameter given by user
        where 'job_id': string, job id parameter chosen by user
        where 'forget_pw_ans': string, forget password answer parameter given by user
        where 'field_of_study': string, field of study parameter given by user
        where 'user': object, user object in database which's token was matched

        returns True if given values appropriate to insert in database and the values correctly matches
with a user in the database, else False

    This function takes input parameters and checks them if they are valid and return the boolean
result.
```

User Deletion

NAME rest_api.delete_user_t.delete_user_f

DESCRIPTION

```
Created on May 23, 2020
This script controls the delete user api of PLATON_API, using django&mysql backend.
Endpoint description:
    http://localhost:8000/api/deleteuser/

    Anything other than 'POST':
        Returns error
    'POST':
        Deletes the logged in user.
        JSON Format : {
                        token = "Token given by the system when you log in",
                        email = "Logged in users email"
                        }
@author: Oyku Yilmaz
@company: Group7
```

CLASSES

```
builtins.object
    DeleteUser

class DeleteUser(builtins.object)
 |  In this class, delete user endpoint is implemented
 |
 |  Static methods defined here:
 |
 |  bestsellers()
 |      returns a message including a book recommendation in the format "NAME OF THE BOOK " by "AUTHOR".
"DESCRIPTION".
 |
 |      This function returns a message including a book recomendation taken by NYT bestseller list.
 |
 |  deleteUser2(request)
 |      where 'request': HTTP request that is from the view class
 |
 |      returns nothing if everything goes right, returns an error message otherwise.
 |
 |      This function deletes the logged in user.
 |
 |  verifyTokenAndMail(token=None, email=None)
 |      where 'token': string, 64 characther string that can be token
 |
 |      where 'email': string
 |
 |      returns TRUE if the token and the email belongs to a specific user in the database
 |
 |      This function verifies if both the mail and token exist in the database.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

Joke of the Day Documentation

NAME rest_api.joke.joke - Created on MAY 22, 2020

DESCRIPTION

```
This script controls the joke api of PLATON_API, using django&mysql backend.
Endpoint description:
    http://localhost:8000/api/joke/

    'GET':
        Returns a joke from https://api.jokes.one/jod api.
        Get token from request header.
        JSON Format : { 'token': "" }   #token of the user


@author: Ertugrul Bulbul, ertbulbul
@company: Group7
```

FUNCTIONS

```
verify_token(token=None)
       where 'token': string, 64 characther string that can be token

       returns True if token exists in the DB

    This function verifies the token if there exits.

joke_api(request)
        where 'response': rest_framework response

        returns response from rest_framework

    This function only accepts GET requests, and valid token is neccessary to return joke


get_joke():

        return a python dictionary, json that includes joke of the day

    This function returns a joke using an external api.
```

Translation Documentation

NAME

```
rest_api.translation.translation
```

DESCRIPTION

```
Created on MAY 18, 2020
This script controls the translate functionality api of PLATON_API, using django mvc model.
Endpoint description:
    http://localhost:8000/api/translate/<str:token>

    'GET':
        Gets token from url.
        token = "Your token will be in url",

@author: Hasan Ramazan Yurt
@company: Group7
```

FUNCTIONS

```
translate(request, token)
    This function returns a translated about me part of user whose token is given by using an external
api.
```

API Documentations

Register API Documentation

DESCRIPTION

This collection of endpoints is designed to give registration authorization. It takes necessary parameters from user and if valid it inserts into database.BELOW YOU CAN FIND AS TEXT FORMAT, BUT FROM
HERE A HTML FORMAT DONT FORGET YOU NEED TO DONWLOAD TO VIEW HTML

Endpoint /api/register

This script controls the registration api of PLATON_API, using django&mysql backend.
http://localhost:8000/api/register/

'GET':
Returns a random poem using an external API as JSON 'POST':
Gets dictionary from body.
JSON Format : {'name': "", string, Name parameter given by user 'surname':
"", string, Surname parameter given byuser
'password1': "", string, Password1 given by user
'password2':"", string, Password2 given by user to checkwhether Password1 ismatched.
'e_mail':"", string, Email parameter given by user
'about_me':"", string, About me parameter given by user
'job_name':"", string, one word job description 'forget_password_ans':"", string, forget password answer parameter given by user
'field_of_study':""} string, field of study parameter given byuser

Example Requests

- POST: "http://localhost:8000/api/register", content-type: "application/json"

```
{
        'name':'burak','surname':'ömür','e_mail':'burak.omur@gmail.com','about_me':'a student only',
'job_name':"engineer",'forget_password_ans':"father" , 'field_of_study':"science",'password1':
"123456",'password2':"123456"
```

- GET: "http://localhost:8000/api/register"

```
{
```

Response Status Codes

- POST:

```
HTTP_201_CREATED : If successfully inserted into database.

HTTP_400_BAD_REQUEST : If given parameters not appropriate to insert into database. HTTP_403_FORBIDDEN: If some
error occurred due to api/server/database.
```

- GET:

```
HTTP_200_OK : Supports get request and returns a random poem in body in application/json format.
```

Endpoint /api/register/fe

```
This script controls the registration api of PLATON_API, using django&mysql backend. http://localhost:8000/api/register/fe
```

```
    'GET':
        Renders a html page that includes form and random poem. 'POST':
        Gets dictionary from body.
        JSON Format : {'name': "",                  string, Name parameter given by user 'surname':
                       "",                          string, Surname parameter given byuser
                       'password1': "",             string, Password1 given by user
                       'password2':"",              string, Password2 given by user to checkwhether
Password1 ismatched.
                       'e_mail':"",                 string, Email parameter given by user
                       'about_me':"",               string, About me parameter given byuser
                       'job_name':"",               string, one word job description 'forget_password_ans':"",
                                                    string, forget password answer parameter given by
user
                       'field_of_study':""}         string, field of study parameter given byuser
```

Using form of our known HTML it takes parameters and posts into above described endpoint. Output:

- Always a random poem.
- Redirects into itself.
- If fail to insert, it shows an error message.

Search API Documentation

This module makes a search in the registered users of the platform. You can give a search string and filters about the job and field of study of the user that you want to find. Also, a sorting criteria can be given as parameter. It searches and finds a list of users according to your criteria.

Input Format

This endpoint has no POST request. So if you send a post request it will give an error. The output is the following:

GET request parameters:

```
{
    token = "Your token will be here!!",
    search_string = "The string that you want to search will be here",
    filter = {"job" = "job_filter", "field_of_study" = "field_of_study_filter"} sorting_criteria = "Sorting criteria will be
}
```

- token parameter is explained in the **Authentication** part.
- search_string parameter can be any string that you want to search. To see all of the users give an empty string (like search_string="" not search_string=) as the value of this parameter. You have to give the search_string parameter.
- filter parameter is set of filters in the system. job can be any job in our system. field_of_study can be any string. filter parameter and its subparameters are not obligatory parameters. If you don' give this as a parameter system makes a search without filtering. sorting_criteria is the sorting criteria of the result list. If any parameter is given the default sorting is made according to the semantic points of the results.
- **name_increasing**, **name_decreasing**, **surname_increasing**, **surname_decreasing** can be given as a parameter. It sorts the result list according to the parameter given.

Authentication

This module uses the tokens of the system for authentication. To use this endpoint, firstly register to the system with `../api/register/` endpoint and get a token using `../api/login/` endpoint. And give this

token as an input to the request as:

```
{
	token = "Your token will behere!!"
}
```

Sample Output

Request

> *GET http://127.0.0.1:8000/api/search/?token=8245147978909677 18&search_string=trial&filter= {"job":"student"}*

Output

```
[
	{
		"name": "Test1",
		"surname": "Test1",
		"e-mail": "test@test.com",
		"about_me": "I am a test user and I will be deleted when API is deployed!!", "job": "Student",
	},
		"name": "Test1",
		"surname": "Test1",
		"e-mail": "test@test.com",
		"about_me": "I am a test user and I will be deleted when API is deployed!!", "job": "Student",
	},
		"name": "Test1",
		"surname": "Test1",
		"e-mail": "test@test.com",
		"about_me": "I am a test user and I will be deleted when API is deployed!!", "job": "Student",
	},
		"name": "Test1",
		"surname": "Test1",
		"e-mail": "test@test.com",
		"about_me": "I am a test user and I will be deleted when API is deployed!!", "job": "Student",
	}
]
```

Error Codes

- Token Error:

> *GET http://127.0.0.1:8000/api/search/?search_string=trial&filter={"job":"student"}*

```
"You have to give your token" Status Code: 401
```

- 3rd Party API Error:

> *GET http://127.0.0.1:8000/api/search/?token=8245147978909677 18&search_string=trial*

```
"There is a problem about the 3rd party APIs that I used. Please try again!!" , Status Code: 500
```

- Input Error:

> *GET http://127.0.0.1:8000/api/search/?token=8245147978909677l8&search_string=trial&filter=*
>
> *{"job":"dent"}*

```
"You give your input in wrong format. Please check the API documentation for the appropriate input format!!" Status Code: 400
```

User Deletion API Documentation

This module deletes the logged in users account from the database. It returns a best seller book recommendation for the user who deleted their account.

Input Format

This endpoint has no GET request. So if you send a get request it will give an error. The output is the following:

POST request parameters:

```
{
        token = "Token given by the system when you log in",

}
```

- `token` parameter is explained in the **Authentication** part. parameter is the e-mail address of
- the logged in user

Authentication

This module uses the tokens of the system for authentication. To use this endpoint, firstly register to the system with ../api/register/ endpoint and get a token using token as an input to the `../api/login/` endpoint. And give this request as:

```
{
        token = "Token given by the system when you log in"
}
```

Sample Output

Request

> *POST http://127.0.0.1:8000/api/deleteuser/ -- with valid token and email*

Output

```
"Not interested in research anymore? Maybe a book can help you refresh yourself.Here is a bestseller book recommendation by New York TimesWALK THE WIRE by David BaldacciThe sixth book in the Memory Man series. Decker and Jamison investigate a murder in a North Dakota town in a fracking boom."
```

Error Codes

- Token Error:

> *POST http://127.0.0.1:8000/api/deleteuser/ -- without token but with valid email*

```
"No token found."
```

-

> *POST http://127.0.0.1:8000/api/deleteuser/ -- without email but with valid token*

- Mail error

```
"No e-mail address found."
```

- Input Error:

> *POST http://127.0.0.1:8000/api/deleteuser/ -- with email and token but they do not belong to a spesific user*

```
"No user found."
```

- Input Error:

> *POST http://127.0.0.1:8000/api/deleteuser/ -- with email and token but something goes wrong*

```
"Cannot retrieve your token or your mail at the moment, sorry."
```

Update User API Documentation

This module allows registered users to update their "name", "surname", "about me", "job name", "forget password answer" information. User needs to enter the correct email address and password to update this information. All information changes are made by filling a single page. Fields left blank are preserved as before.

Input Format

This endpoint has no GET request. So if you send a post request it will give an error. The output is the following:

POST request parameters:

```
{
    token = "Token given by the system when user logged in", name = "Logged in
    user's name",

    surname = "Logged in user's surname", password1 =
    "Logged in user's password", e_mail = "Logged in user's
    email",

    about_me = "Logged in user's about-me part", job_name =
    "Logged in user's jobs",
}
```

- token parameter is explained in the **Authentication** part.
- name parameter is a string. It is the saved name given by the registered user during registration.
- surname parameter is a string. It is the saved surname given by the registered user during registration.
- password1 parameter is a string. It is the saved password given by the registered user during registration.
- e_mail parameter is a string. It is the saved e-mail given by the registered user during registration.
- about_me parameter is a string. It is the saved about-me part given by the registered user during registration.
- job_name parameter is a string. It is the saved job name given by the registered user during registration.
- forgot_pw_ans parameter is a string. It is the saved forget password answer given by the registered user during registration.

- field_of_study parameter is a string. It is the saved field_of_study given by the registered user during registration.

Authentication

This module uses the tokens of the system for authentication. To use this endpoint, firstly register to the system with ../api/updateUser/ endpoint and get a token using this token as an input to the ../api/login/ endpoint. And give request as:

```
{
    token = "Token given by the system when user logged in"
}
```

Sample Output

Request

> *POST http://127.0.0.1:8000/api/updateUser/ -- with valid password and email.*

Output

```
"User information is updated!" Status Code: 201
```

Error Codes

- Mail & Password Error:

> *POST http://127.0.0.1:8000/api/updateUser/ -- with wrong password or email.*

```
"Not valid input. Check password or give valid inputs to the fields!", Code Status: 403
```

- Password & Mail Mismatch Error:

> *POST http://127.0.0.1:8000/api/updateUser/ -- with a password that does not match the email given.*

```
"Not valid input. Check password or give valid inputs to the fields!", Code Status: 403
```

- 3rd Party API Error:

> *POST http://127.0.0.1:8000/api/updateUser/ with valid input or not valid input*

```
"Error! Information could not be changed." Code Status: 403
```

Daily News API Documentation

This API provides user specific news functionality. It checks user's field of study then provide recommended articles for user

Input Format

This endpoint has GET request.

```
{
    token = "Token given by the system when you log in",
}
```

This endpoint has no POST request. So if you send a get request it will give an error. The output is the following:

```
{
    "content": "BadRequest"
}
```

- token parameter is explained in the **Authentication** part.

Authentication

This module uses the tokens of the system for authentication. To use this endpoint, firstly register to the system with ../api/register/ endpoint and get a token using token as an input to the `../api/login/` endpoint. And give this request as:

```
{
    token = "Token given by the system when you log in"
}
```

Sample Output

Request

> *GET http://127.0.0.1:8000/api/news/ -- with valid token*

Example Output

{'url': ['http://www.bbc.co.uk/news/uk-politics-52800595', 'http://www.bbc.co.uk/news/business- 52800611', 'http://www.bbc.co.uk/news/world-latin-america-52796519', 'http://www.bbc.co.uk/news/world- europe-52796699', 'http://www.bbc.co.uk/news/entertainment-arts-52796964', 'http://www.bbc.co.uk/news/business-52795376', 'http://www.bbc.co.uk/news/world-us-canada-52795447', 'http://www.bbc.co.uk/news/world-asia-52794434', 'http://www.bbc.co.uk/news/live/world-52793960', 'http://www.bbc.co.uk/news/world-middle-east-52790864'], 'imageUrl': ['https://ichef.bbci.co.uk/images/ic/1024x576/p08f4d8g.jpg',
'https://ichef.bbci.co.uk/news/1024/branded_news/A40B/production/_112459914_reuters.jpg',
'https://ichef.bbci.co.uk/news/1024/branded_news/2A15/production/_112437701_gettyimages-980341738.jpg',
'https://ichef.bbci.co.uk/news/1024/branded_news/14F7B/production/_112438858_mediaitem112438857.jpg',
'https://ichef.bbci.co.uk/news/1024/branded_news/77DB/production/_112438603_pa-28516614.jpg',
'https://ichef.bbci.co.uk/news/1024/branded_news/05F6/production/_101162510_gettyimages-109422673.jpg',
'https://ichef.bbci.co.uk/news/1024/branded_news/E8B5/production/_112437595_b6cc5fa4-328c-4b0a-b4c3- 456f42316541.jpg',

'https://ichef.bbci.co.uk/news/1024/branded_news/10723/production/_112436376_p08f3b9b.jpg', 'https://m.files.bbci.co.uk/modules/bbc-morph-news-waf-page-meta/4.1.2/bbc_news_logo.png', 'https://ichef.bbci.co.uk/news/1024/branded_news/95AD/production/_111171383_hi054737257.jpg'], 'title': ['"I don't regret what I did,' says Cummings", 'Spain to stop quarantining arrivals from 1 July', 'Where a convicted murderer could win re-election', 'Fancy a pint? Five ways Europe is easing lockdown', "Brian May 'could have died' after heart attack", "Volkswagen loses landmark German 'dieselgate' case", 'Americans flock to beaches on Memorial Day weekend', 'NZ earthquake hits during PM Ardern TV interview', 'Coronavirus updates: US bars travellers from Brazil as outbreak worsens', "Top exiled Saudi officer 'targeted though family'"], 'token': '5b764b7b3b9d36005f2a9e1d88ffa24ee5ffa56dd3661fb20e6c3e5f277c8999'}

Error Codes

- Token Error:

> *GET http://127.0.0.1:8000/api/news/ -- with no valid token*

```
"Unauthorized"
```

> *POST http://127.0.0.1:8000/api/news/ --with no valid request*

```
"Bad Request"
```

Joke of the Day API Documentation

This module returns the joke of the day from an external api. It takes token and check if the token is valid or not then if valid it returns the joke.

Input Format

This endpoint has no POST request. So if you send a post request it will give an error. The output is the following:

GET request parameters:

```
{
        token = "Token ofuser",

}
```

- `token` parameter is explained in the **Authentication** part.

Authentication

This module uses the tokens of the system for authentication. To use this endpoint, firstly register to the system with ../api/register/ endpoint and get a token using token as an input to the `../api/login/` endpoint. And give this request as:

```
{
        token = "Token ofuser"
}
```

Sample Output

Request

> *GET* *http://127.0.0.1:8000/api/joke/?token=824514797890967718*

Output

```
{
        "status": "success",

        "token": "824514797890967718",

        "title": "Courtship Signals".

}
```

Error Codes

- Token Error:

> *GET* *http://127.0.0.1:8000/api/joke/*

```
{
        "status":"invalid_token",

}
```

Forgot Password API Documentation

This module for used to change password if you forget your password. You have to write your e-mail and answer correctly the secret question query. After that you can create your new password.

Input Format

This endpoint has no GET request. So if you send a post request it will give an error. The output is the following:

```
{
    "detail": "Method \"GET\" not allowed."
}
```

POST request parameters:

```
{
    e_mail = "E-mail parameter given by user",


}
```

- `e_mail` parameter is the e-mail address of the user
- `forgot_password_ans` is the secret question query answer(Who is your favourite teacher?)

Sample Output

Request

> *POST http://127.0.0.1:8000/api/forgotpassword/ -- with valid e-mail*

Output

```
[
    {"Your password is changed"}

    {"Passwords did not match"}

    {"Your answer for your registered secret question is false"}

]
```

Error Codes

- Mail Error:

> *POST http://127.0.0.1:8000/api/forgotpassword/ -- with invalid e-mail*

```
"404 pages not found"
```

- Secret Question Answer Error:

> *POST http://127.0.0.1:8000/api/forgotpassword/ -- with valid e-mail and wrong secret question answer*

```
"Your answer for your registered secret question is false"
```

- Incompatible Password Error:

> *POST http://127.0.0.1:8000/api/forgotpassword/ -- with valid e-mail and secret question answer but incompatible password*

```
"Passwords did not match"
```

Translation Documentation

This module translates about me part of a registered user according to the given token. The translation is from English to Yoda.

Input Format

This endpoint has no POST request. So if you send a post request it will give an error. The output is the following:

```
{
    "detail": "Method \"POST\" not allowed."
}
```

GET request parameters:

```
{
    token = "Your token will be in URL",
}
```

- `token` parameter is explained in the **Authentication** part.

Authentication

This module uses the tokens of the system for authentication. To use this endpoint, firstly register to the system with `../api/register/` endpoint and get a token using token as an input to the `../api/login/` endpoint. And give this request as:

```
{
        token = "Your token will be in the URL"
}
```

Sample Output

Request

> *GET*
>
> *http://127.0.0.1:8000/api/translation/b2a52d9e497071389f750333b4376e920d6e6c80b6fa84a43c4684ee614f5ee9*

Output

```
[
   {
"translated": "A junior student at bogazici university and my major degree is in computer engineering, I am,Also I study a minor degree in electrics and electronics engineering.Finally,I like everything about basketball."
   }
```

Error Codes

- Invalid Token Error: When one input invalid token

> *GET http://127.0.0.1:8000/api/translation/invalidtoken*

```
"error": "Unauthorized access" Status Code: 404
```

- Key Error: When one requests more than 5 in an hour.

> *GET*
>
> *http://127.0.0.1:8000/api/translation/b2a52d9e497071389f750333b4376e920d6e6c80b6fa84a43c4684ee614f5ee9*

```
{
"error": {
        "code": 429,
        "message": "Too Many Requests: Rate limit of 5 requests per hour exceeded. Please wait for 59 minutes and 56 seconds."
                }
```

Login Documentation

This module allows a registered user to log in to the platform by returning the user a unique token.

Input Format

This endpoint has no POST request. So if you send a post request it will give an error. The output is the following:

GET request parameters:

```
{
        e_mail = "Your email will be here.",

}
```

- `e_mail` parameter is the e-mail address user provides while registering to the system via `../api/register/` endpoint and then uses everytime they want to login to the platform. It must be a string in a e-mail address form, as indicated in the implementation.
- `password` parameter is the password users set while registering to the to the system via `../api/register/` endpoint and can be changed via `../api/forgot_password/` . It is hashed for extra protection.

Sample Output

Request

> *GET http://127.0.0.1:8000/api/login/? e_mail=dummytest@hotmail.com&password=677g43e38y9608hs327604gp49j69*

Output

```
"Welcome back!"
```

Error Codes

- Wrong Password:

> *GET http://127.0.0.1:8000/api/login/? e_mail=dummytest@hotmail.com&password=38qlm2t1r22sue3oytkt7p8hti38m*

```
"Invalid e-mail or password!"
```

- Incorrect Email Format:

> *GET http://127.0.0.1:8000/api/login/? e_mail=dummytesthotmail.com&password=677g43e38y9608hs327604gp49j69*

```
"Please use a valid e-mail address."
```

- Empty Input:

> *GET http://127.0.0.1:8000/api/login/?e_mail=dummytest@hotmail.com&password=*

```
"Please fill all the required fields."
```

Logout Documentation

When a user sends a request to this endpoint, the user gets logged out from the app.

URL

/api/logout/?token=[user_session_token]

- *Method:*

  `GET`

- **URL Params**

  Required:

  `token=[string]`

- **Success Response:**

  - **Code:** 200, **Content:** `"You have succesfully logged out. Goodbye, see you soon!"`

- *Error Response:*

  - **Code:** 401 Unauthorized, **Content:** `"401 Unauthorized"`

  OR

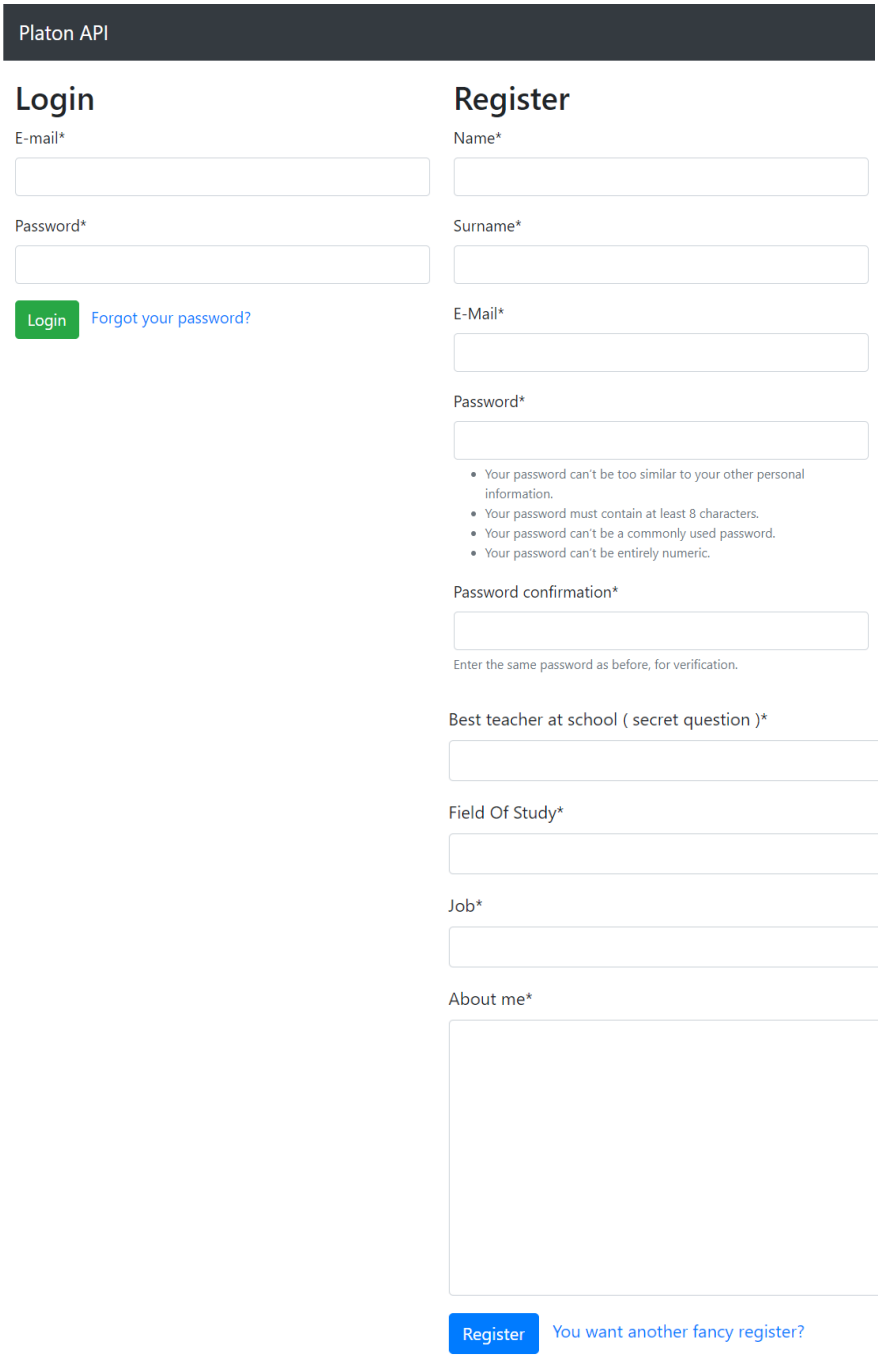
  - **Code:** 400 Bad Request, **Content:** `"400 Bad Request"`

- *Sample Call:*

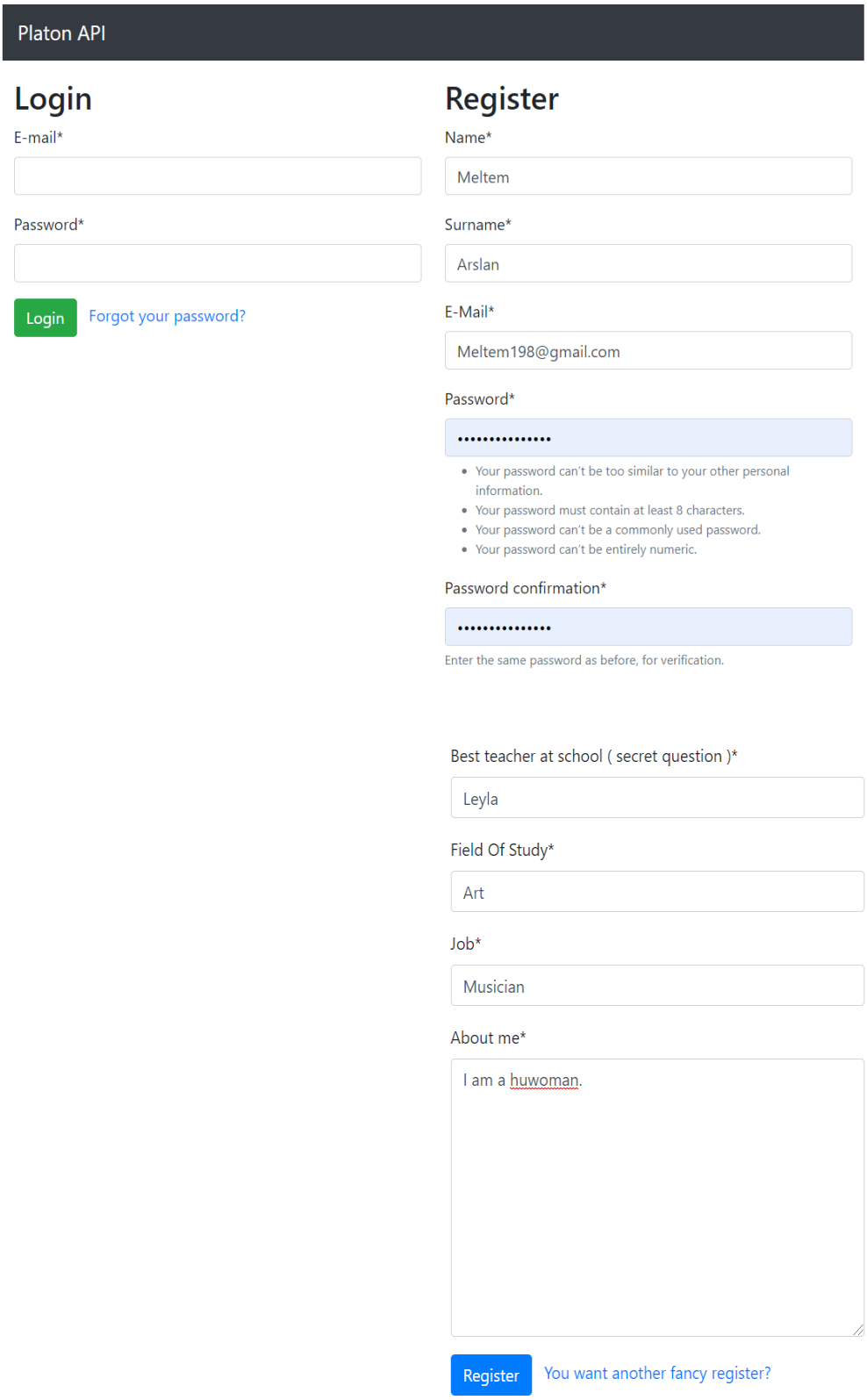
  curl http://127.0.0.1:8000/api/logout/? token=7dec2beca7530a60223035e5928bc00e68aad71b2288c9afb32784a70ce4e518

Practice-app

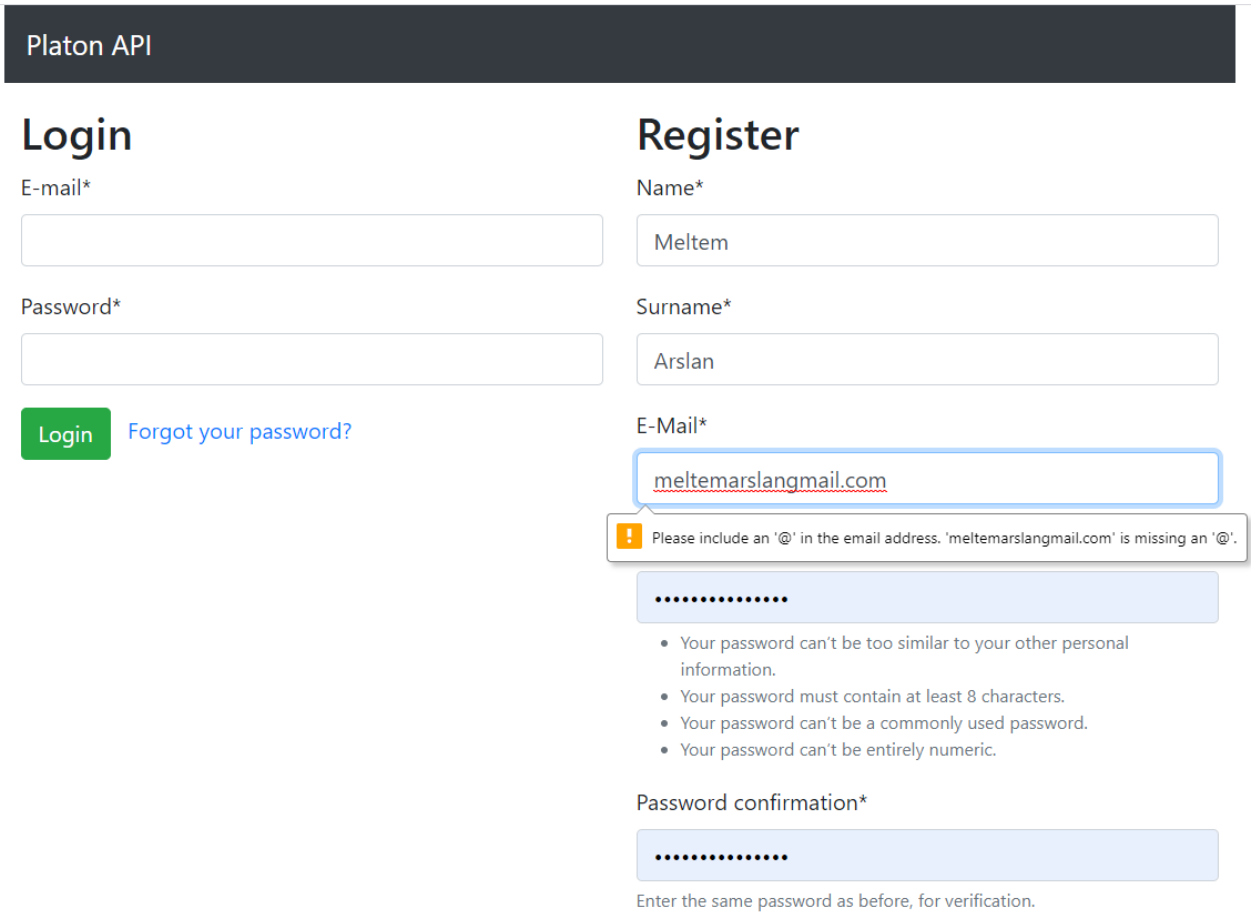
1- LOGIN & REGISTRATION PAGE:



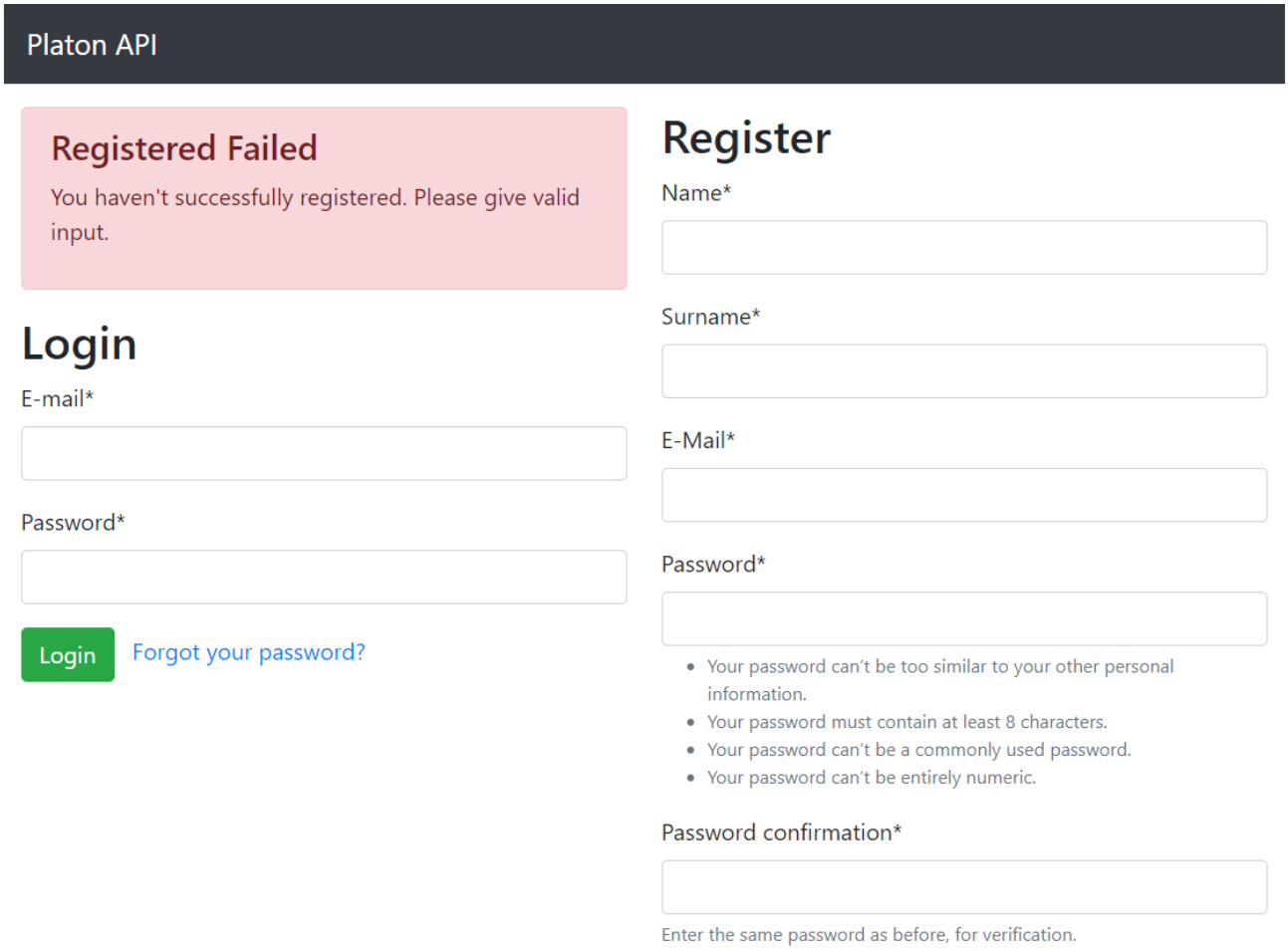
2- LOGIN & REGISTRATION PAGE - WHILE REGISTERING:



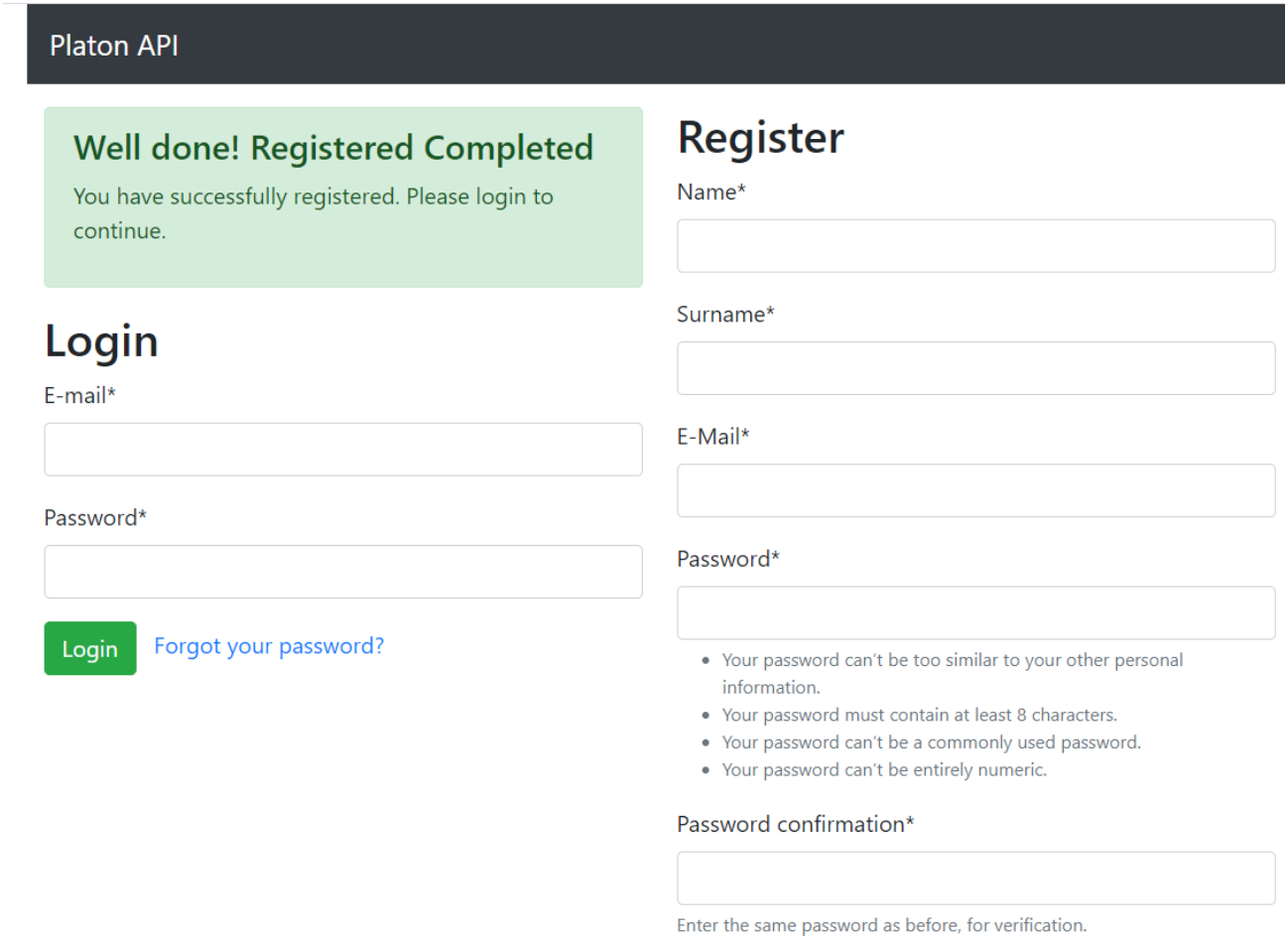
3- LOGIN & REGISTRATION PAGE - UNVALID MAIL :



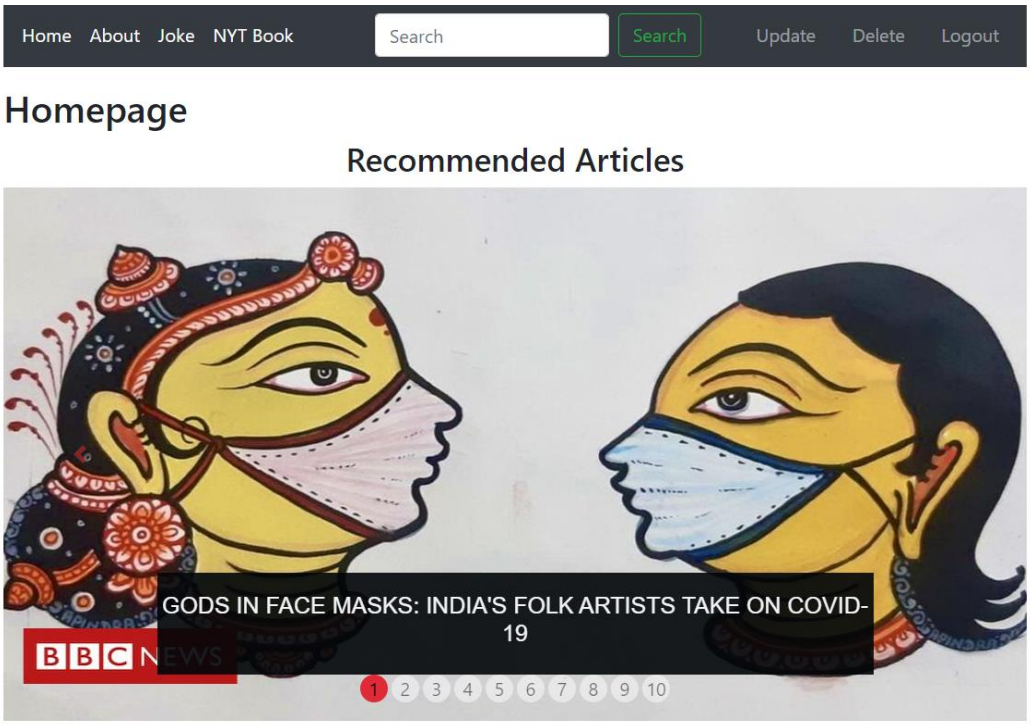
4- LOGIN & REGISTRATION PAGE - FAILED REGISTRATION:

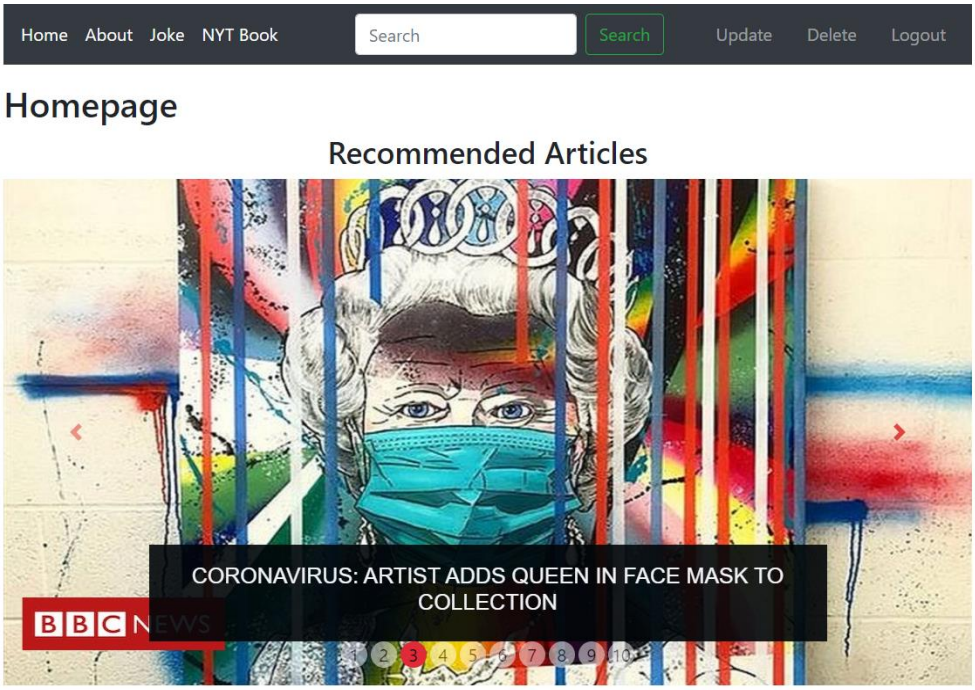


5- LOGIN & REGISTRATION PAGE - SUCCESSFUL REGISTRATION:

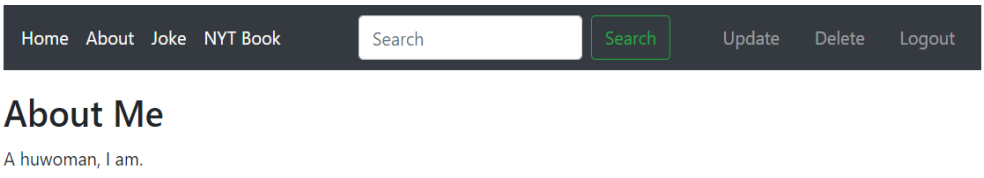


6- HOMEPAGE:

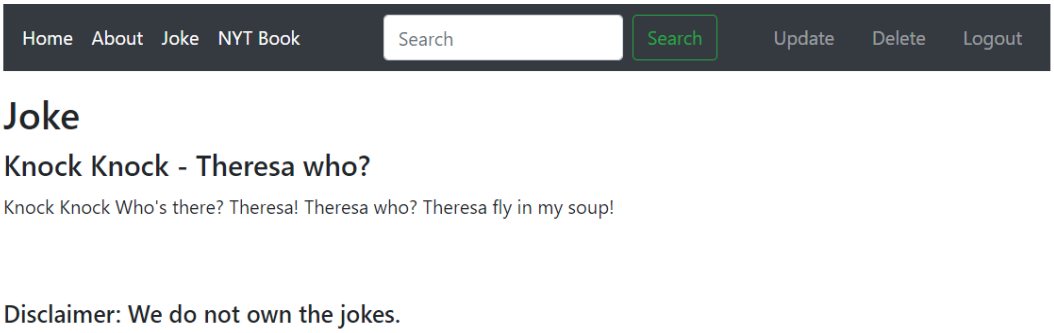

Home About Joke NYT Book
Search
Search
Update Delete Logout
Homepage
Recommended Articles
GODS IN FACE MASKS: INDIA'S FOLK ARTISTS TAKE ON COVID-19
BBC NEWS
1 2 3 4 5 6 7 8 9 10


Home About Joke NYT Book
Search
Search
Update Delete Logout
Homepage
Recommended Articles
CORONAVIRUS: ARTIST ADDS QUEEN IN FACE MASK TO COLLECTION
BBC NEWS
1 2 3 4 5 6 7 8 9 10

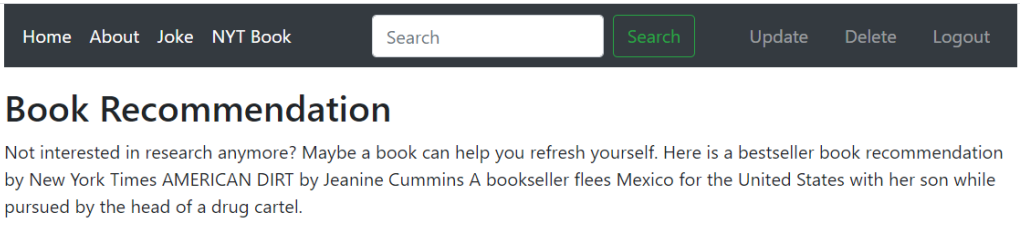
7- ABOUT ME PAGE:



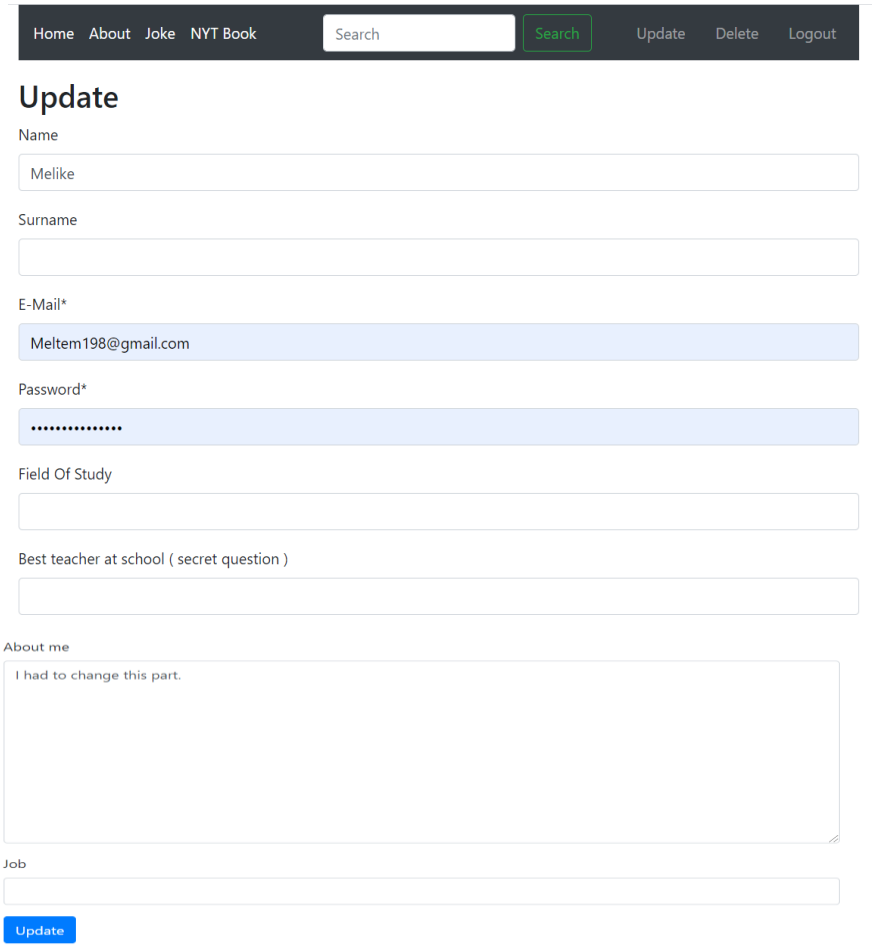
8- JOKE PAGE:



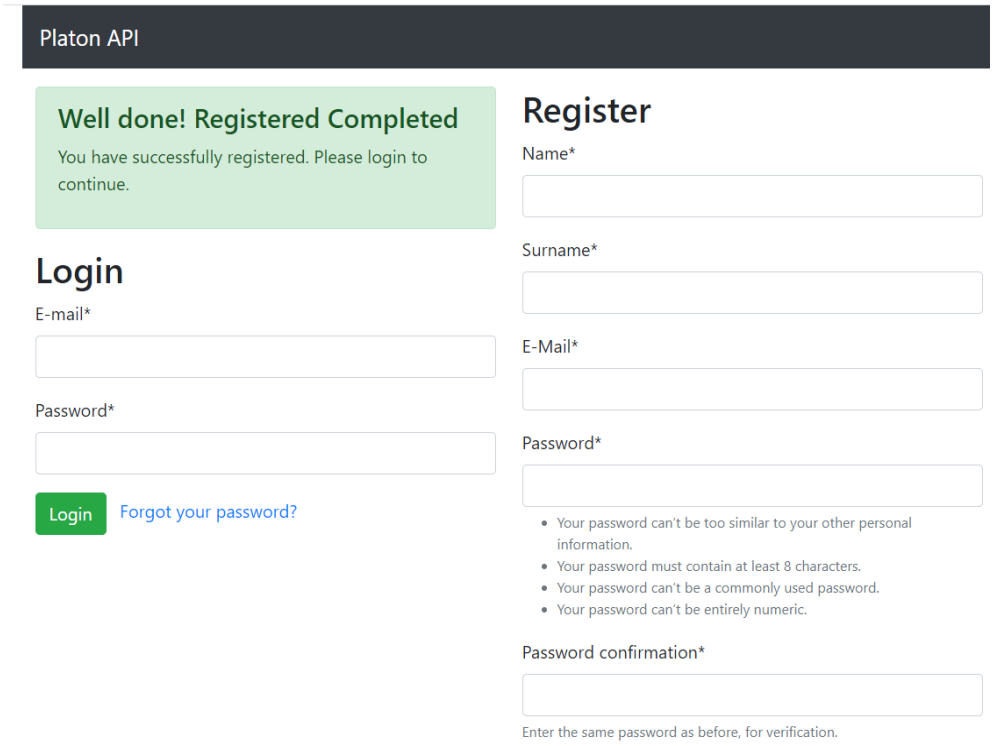
9- NYT Book PAGE:



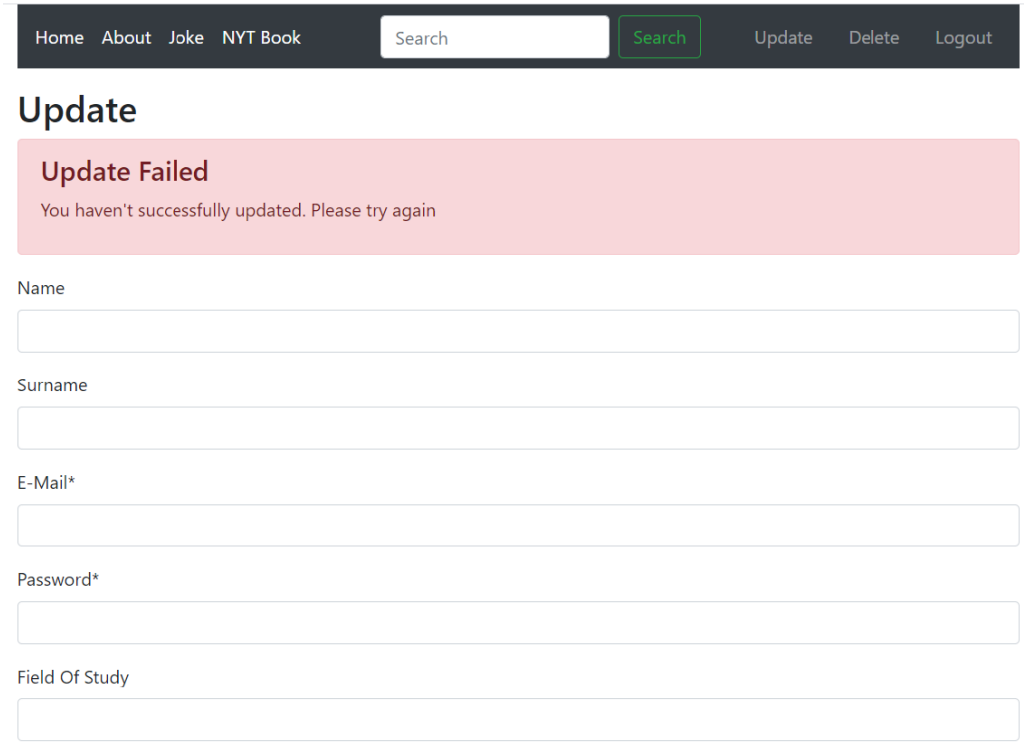
10- UPDATE PAGE -FILLED-:



11- SUCCESSFUL UPDATE:



12- FAILED UPDATE:



13- SEARCH RESULTS: