Candidate Management System

Installation Guide To Run The Project **2**

Problem Statement **4**

Functional Requirements: **4**

System Flow **4**

Endpoints & How To Work With Login / Signup & CRUD APIs **5**

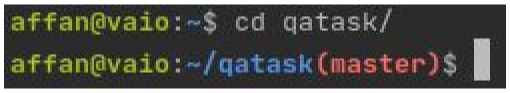
Table Of User Actions to Endpoint Mapping 5
User Actions to Endpoint Mapping Detail 6
Details To Sign Up 6
Details To Login 7
Details To Get List Of All Candidates 8
Details To View A Specific Candidate’s Detail 10
Details To Add New Candidate 11
Details To Update a Candidate 13
Details To Delete a Candidate 14
Details To Patch A Candidate 16

Installation Guide To Run The Project

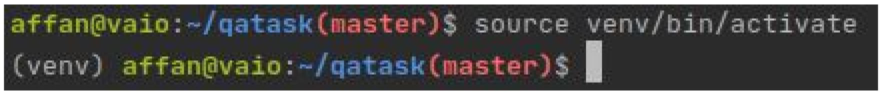
NOTE: The project was built in Ubuntu 18.04 so installation complies with it as well as MacOS.

After downloading the project, perform the following operations to run the system:

1. Ensure you have **Python 3.6+** or **Python 3.6.9**, **pip** and **virtualenv** installed.
2. **cd** into the **qatask** folder using the command **cd qatask/** as follows:

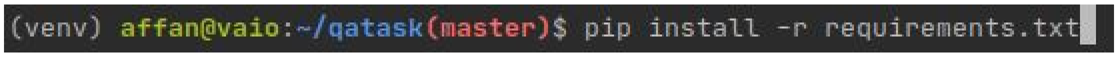
   

   ```
   affan@vaio:~$ cd qatask/
   affan@vaio:~/qatask(master)$
   ```
3. Now type in **source venv/bin/activate** and hit **Enter** to activate the virtual environment.

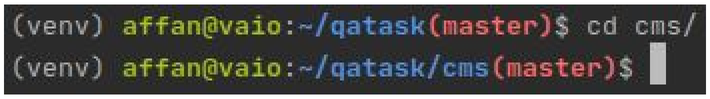
   

   ```
   affan@vaio:~/qatask(master)$ source venv/bin/activate
   (venv) affan@vaio:~/qatask(master)$
   ```

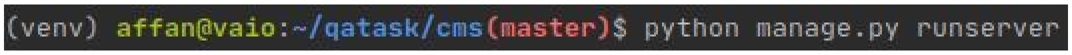
   Note that the virtual environment has been activated with **venv** appearing before the system name.
4. Now type in **pip install -r requirements.txt** to install all dependent packages via pip and hit **Enter**:

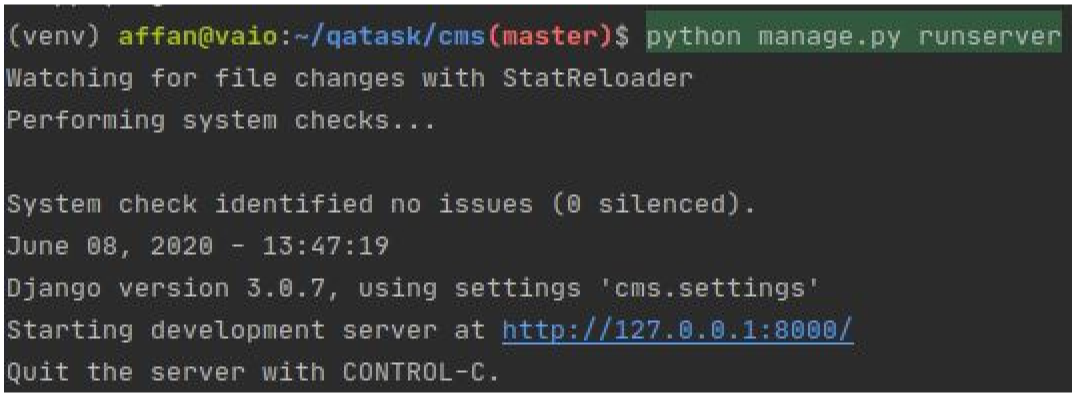
   ```
   (venv) affan@vaio:~/qatask(master)$ pip install -r requirements.txt
   ```
5. Wait for all packages to download.
6. After all packages have been downloaded, **cd** into the **cms** folder using the command **cd cms/** and hit Enter:

   ```
   (venv) affan@vaio:~/qatask(master)$ cd cms/
   (venv) affan@vaio:~/qatask/cms(master)$
   ```
7. Now type in **python manage.py runserver** to run the server:

   ```
   (venv) affan@vaio:~/qatask/cms(master)$ python manage.py runserver
   ```
8. Access the website using 127.0.0.1:8000/ in your browser or you could directly access it through the URL here after running server:



```
(venv) affan@vaio:~/qatask/cms(master)$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 08, 2020 - 13:47:19
Django version 3.0.7, using settings 'cms.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

9. Now you can play around with the system and use clients like Postman to access the REST APIs.

Problem Statement

Keep a track of all candidates that have applied for an IT job.

Functional Requirements:

1. For admin users who can login / signup:
    a. See a list of all candidates. Information to store about a candidate is as follows. I went with few attributes for the sake of this task:
        i. Name
        ii. Email
        iii. Contact
        iv. Job applied to
    b. Create a new candidate.
    c. Update a candidate.
    d. Delete a candidate.
    e. Use a filter on the UI that allows them to filter candidates based on data in the resumes.
2. CRUD / PATCH REST APIs for the candidate.
3. Allow candidates to only add information to the portal.

System Flow

1. The landing page (**home page**) is the page where a candidate can ***upload his information*** to apply for a job position. The positions have been ***added to the model*** just as extra information.
2. On clicking upload, the candidate is redirected to a success page from where he can choose to return back to the home page.
3. ***To access the admin side***, you'll have to ***signup*** and ***login*** via the buttons on the top-right corner of the screen. For the sake of this task, any user can signup to the website using the ***Signup Button*** in the navbar and become an admin user but in a production environment, the user who will be signing-up will first have to pass an approval process or the login / signup will be completely restricted from public view.
4. After signing-up, the user is redirected to the admin side of the portal where he / she can CRUD candidates.
5. The ability to search and add candidates is ***only available when a user is authenticated*** i.e. is an admin user who signed up.

Endpoints & How To Work With Login / Signup & CRUD APIs

In order to perform CRUD operations, one must perform a ***signup***. Endpoints to permissible user actions mapping is as follows. Detailed user instructions on how / what parameters to send considering you're using Postman like clients to test the APIs follows the table below:

Table Of User Actions to Endpoint Mapping

Action	**Request**	**Endpoint**
Sign-up	POST	http://127.0.0.1:8000/api/accounts/signup/
Log-in	POST	http://127.0.0.1:8000/api/accounts/login/
See a list of all candidates	GET	http://127.0.0.1:8000/api/candidates/
See a particular candidate	GET	http://127.0.0.1:8000/api/candidates/<id of candidate>/
Add a candidate	POST	http://127.0.0.1:8000/api/candidates/
Update a candidate	PUT	http://127.0.0.1:8000/api/candidates/<id of candidate>/update/
Delete a candidate	DELETE	http://127.0.0.1:8000/api/candidates/<id of candidate>/delete/
Patch a candidate	PATCH	http://127.0.0.1:8000/api/candidates/<id of candidate>/patch/

User Actions to Endpoint Mapping Detail

Details To Sign Up

Endpoint → http://127.0.0.1:8000/api/accounts/signup/
Request type → POST

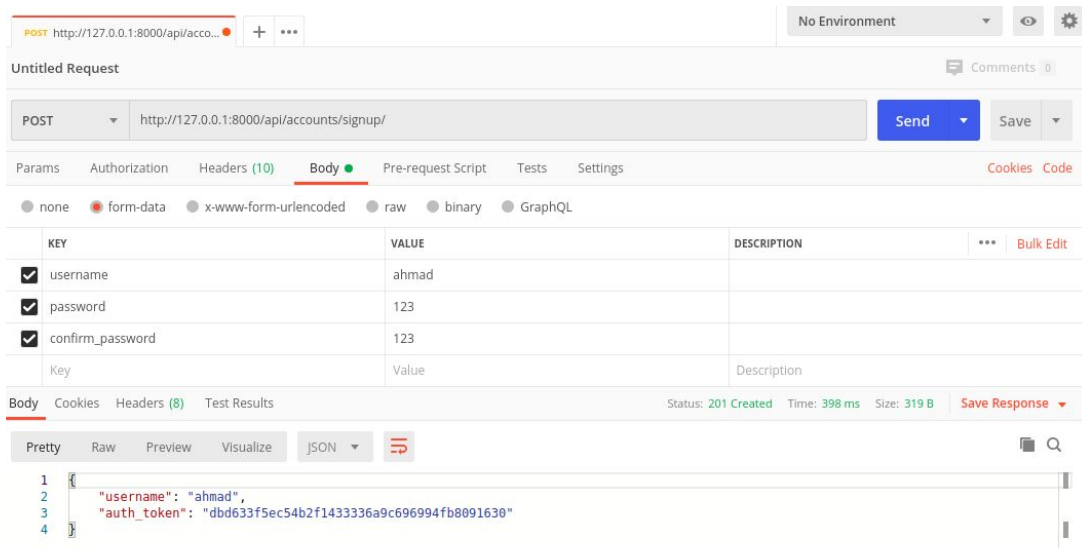
Steps to send data. Please send the Keys as is:

1. Click on **Body**.
2. Check the **form-data** radio button and enter Key / Value as follows. The Keys must be the same as is specified below:

KEY	**VALUE**
username	any name without spaces.
password	Any password.
confirm_password	Matching as above.

   NOTE: The passwords must match otherwise appropriate errors will be returned. All three are required fields.
3. Click **Send**. You'll receive a JSON response specifying the newly created username and your authentication token which will be required to login. ***Please copy it with you***.

End Result:



POST http://127.0.0.1:8000/api/accounts/signup/

	KEY	VALUE	DESCRIPTION
✓	username	ahmad	
✓	password	123	
✓	confirm_password	123	
	Key	Value	Description

Status: 201 Created Time: 398 ms Size: 319 B

```
{
    "username": "ahmad",
    "auth_token": "dbd633f5ec54b2f1433336a9c696994fb8091630"
}
```

Details To Login

Use the following information to ***login***:
Endpoint → http://127.0.0.1:8000/api/accounts/login/
Request type → POST

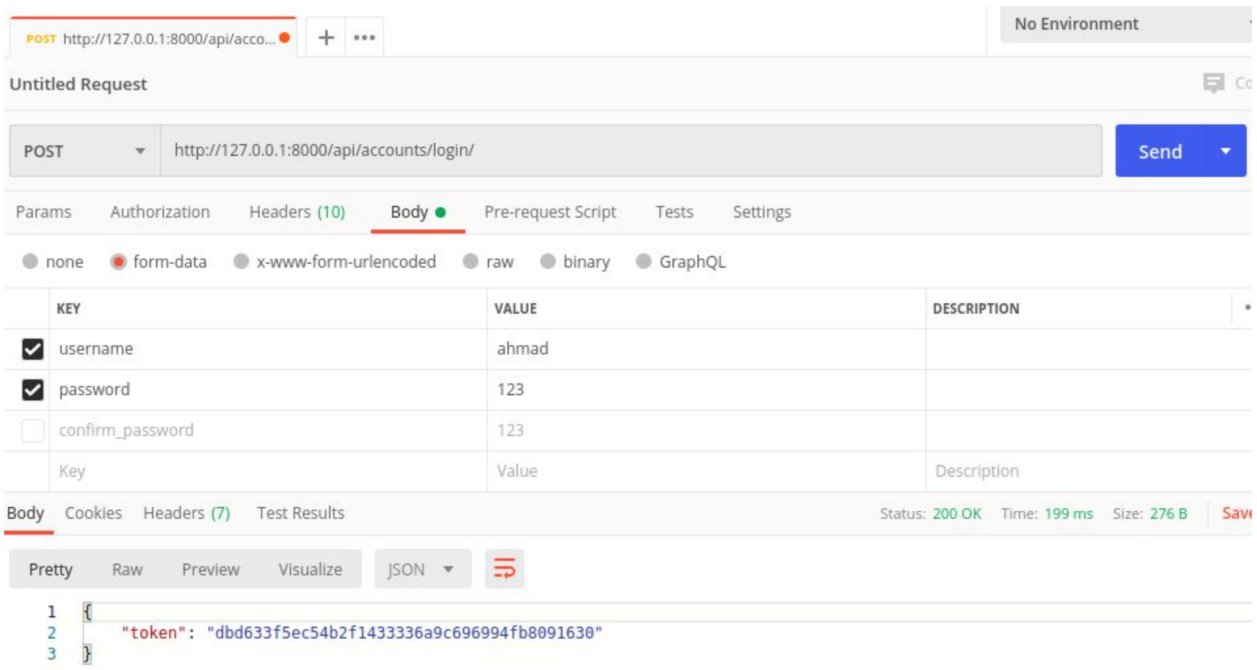
Steps to send data. Please send the Keys as is:

1. Click on **Body**.
2. Check the **form-data** radio button.
3. Enter Key / Value pairs as follows. The Keys must be specified as is:

KEY	**VALUE**
username	Registered username
password	Registered password

4. Now click **Send**. You'll Receive **HTTP Status 200 OK** as well as your **authentication token** back on successful login. You'll need this token to perform any CRUD / PATCH operations.

End Result:



KEY	VALUE	DESCRIPTION
✓ username	ahmad	
✓ password	123	
confirm_password	123	
Key	Value	Description

```
{
    "token": "dbd633f5ec54b2f1433336a9c696994fb8091630"
}
```

Details To Get List Of All Candidates

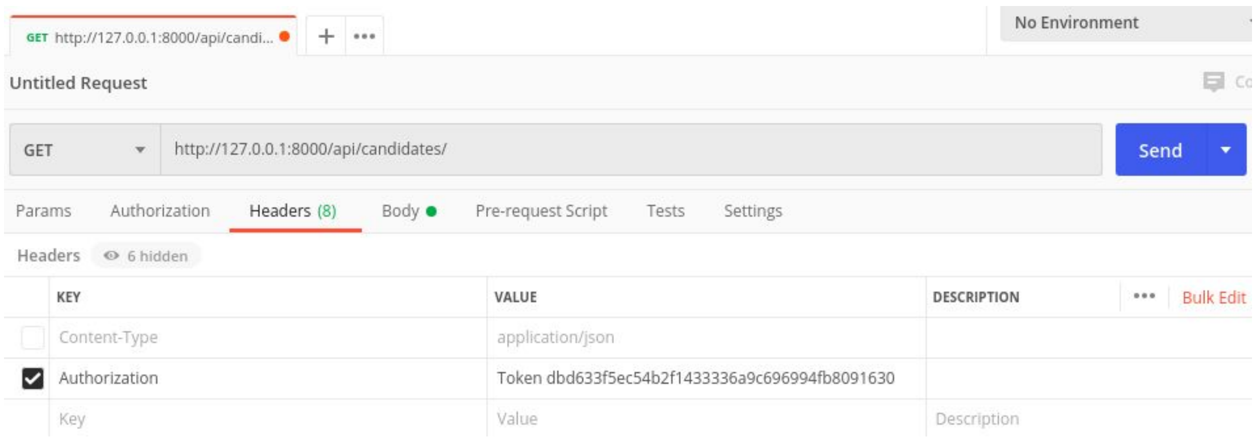
Once logged-in, the user can now perform all CRUD operations. ***To see a list of all candidates***, use the following information:

Endpoint → http://127.0.0.1:8000/api/candidates/
Request type → GET

Steps to send data. Please send the Key(s) as is:

1. Click **Headers**.
2. Check the **form-data** radio button.
3. Enter Key / Value pair as follows:

KEY	VALUE
Authorization	Token <auth_token obtained from signup> **NOTE:** The format contains the keyword **Token** then a **single space** and then the auth_token itself obtained when you signed up.



GET http://127.0.0.1:8000/api/candi...

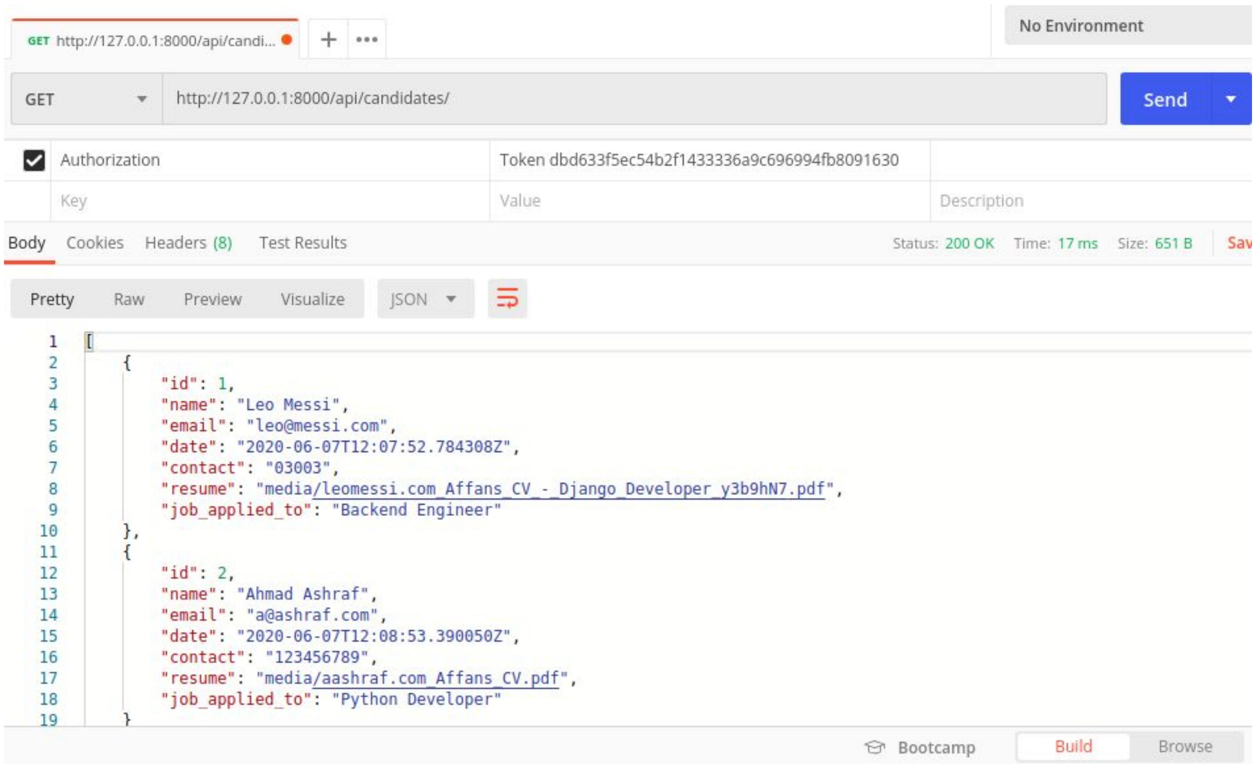
Untitled Request

GET http://127.0.0.1:8000/api/candidates/

KEY	VALUE	DESCRIPTION
Content-Type	application/json	
✓ Authorization	Token dbd633f5ec54b2f1433336a9c696994fb8091630	
Key	Value	Description

4. Click **Send**. You'll Receive **HTTP Status 200 OK** and a JSON Response of all registered Candidates.

End Result:



GET http://127.0.0.1:8000/api/candi...

No Environment

GET http://127.0.0.1:8000/api/candidates/

Send

✓ Authorization	Token dbd633f5ec54b2f1433336a9c696994fb8091630	
Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 17 ms Size: 651 B

Pretty Raw Preview Visualize JSON

```
[
    {
        "id": 1,
        "name": "Leo Messi",
        "email": "leo@messi.com",
        "date": "2020-06-07T12:07:52.784308Z",
        "contact": "03003",
        "resume": "media/leomessi.com_Affans_CV_-_Django_Developer_y3b9hN7.pdf",
        "job_applied_to": "Backend Engineer"
    },
    {
        "id": 2,
        "name": "Ahmad Ashraf",
        "email": "a@ashraf.com",
        "date": "2020-06-07T12:08:53.390050Z",
        "contact": "123456789",
        "resume": "media/aashraf.com_Affans_CV.pdf",
        "job_applied_to": "Python Developer"
    }
```

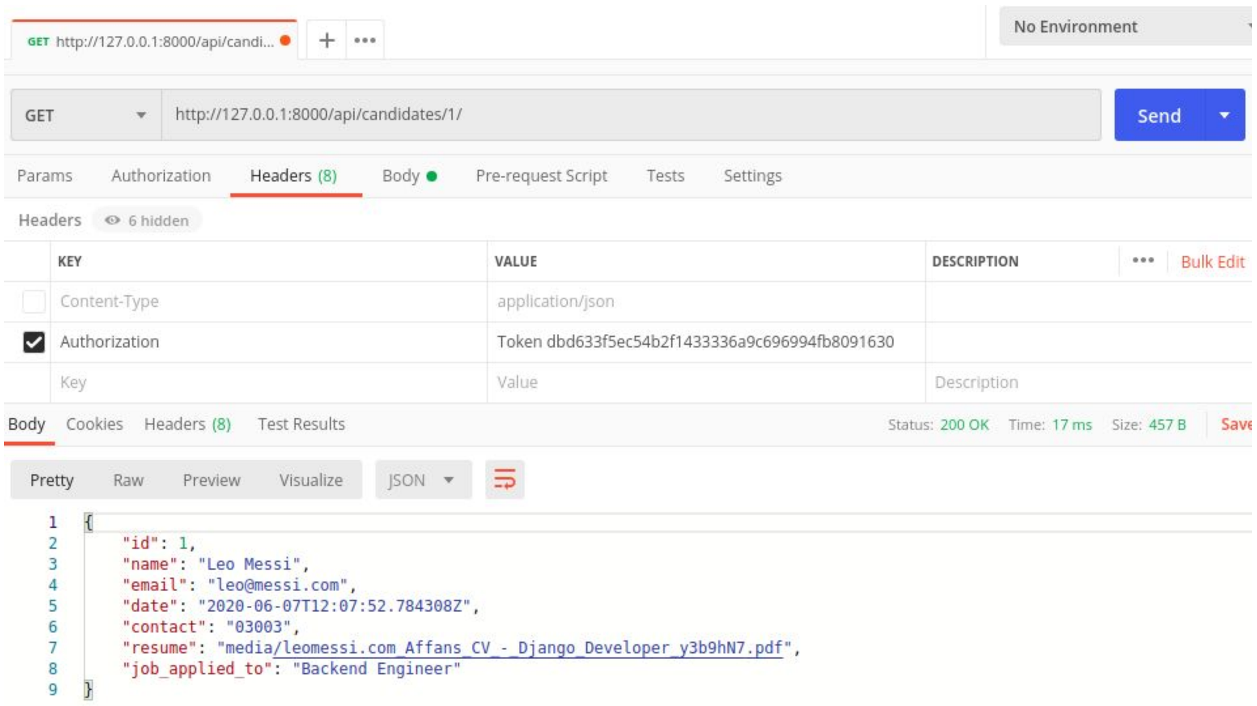
Details To View A Specific Candidate’s Detail

To view details of a specific candidate, use the following information:

Endpoint → http://127.0.0.1:8000/api/candidates/<id of candidate>/
Request type → GET

Use exact same steps from the above process except with a changed endpoint that has an ***Id of a candidate*** with a ***forward slash appended at the end***.

End Result:



GET http://127.0.0.1:8000/api/candidates/1/

	KEY	VALUE	DESCRIPTION
☐	Content-Type	application/json	
✓	Authorization	Token dbd633f5ec54b2f1433336a9c696994fb8091630	
	Key	Value	Description

Status: 200 OK Time: 17 ms Size: 457 B

```
{
    "id": 1,
    "name": "Leo Messi",
    "email": "leo@messi.com",
    "date": "2020-06-07T12:07:52.784308Z",
    "contact": "03003",
    "resume": "media/leomessi.com_Affans_CV_-_Django_Developer_y3b9hN7.pdf",
    "job_applied_to": "Backend Engineer"
}
```

Details To Add New Candidate

To **add a candidate**, use the following information:

Endpoint → http://127.0.0.1:8000/api/candidates/
Request type → POST

Steps to send data. Please send the Keys as is:

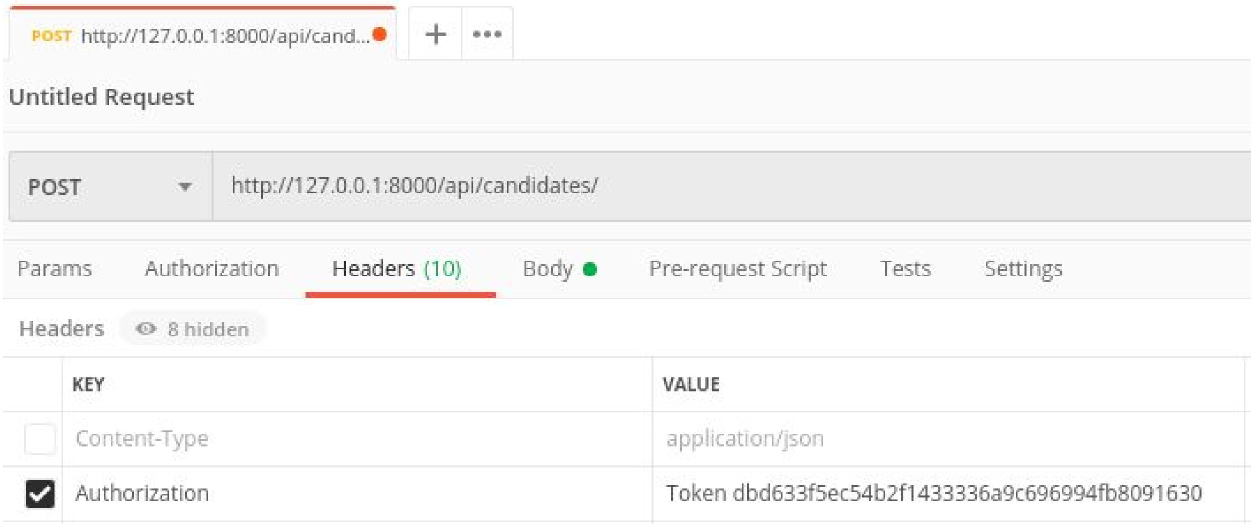
1. Click on **Body**.
2. Check the **form-data** radio button.
3. Enter Key / Value pairs as follows. Keys must be specified as is:

KEY	**VALUE**
name	Name of candidate. Spaces allowed.
email	Valid email address.
contact	Contact of candidate.
resume	Any docx or pdf file.
job_applied_to	Any value (same as is) from the following list. The list is hardcoded in models for now for the sake of this task. Any other values will not be accepted: Backend Engineer Frontend Engineer Python Developer Business Analyst ML Engineer Data Engineer Technical Recruiter

4. Click on **Headers**.
5. Enter Key / Value pair as follows:

KEY	**VALUE**
Authorization	Token <auth_token obtained from

	signup> **NOTE:** The format contains the keyword **Token** then a **single space** and then the auth_token itself obtained when you signed up.


POST http://127.0.0.1:8000/api/cand...

Untitled Request

POST http://127.0.0.1:8000/api/candidates/

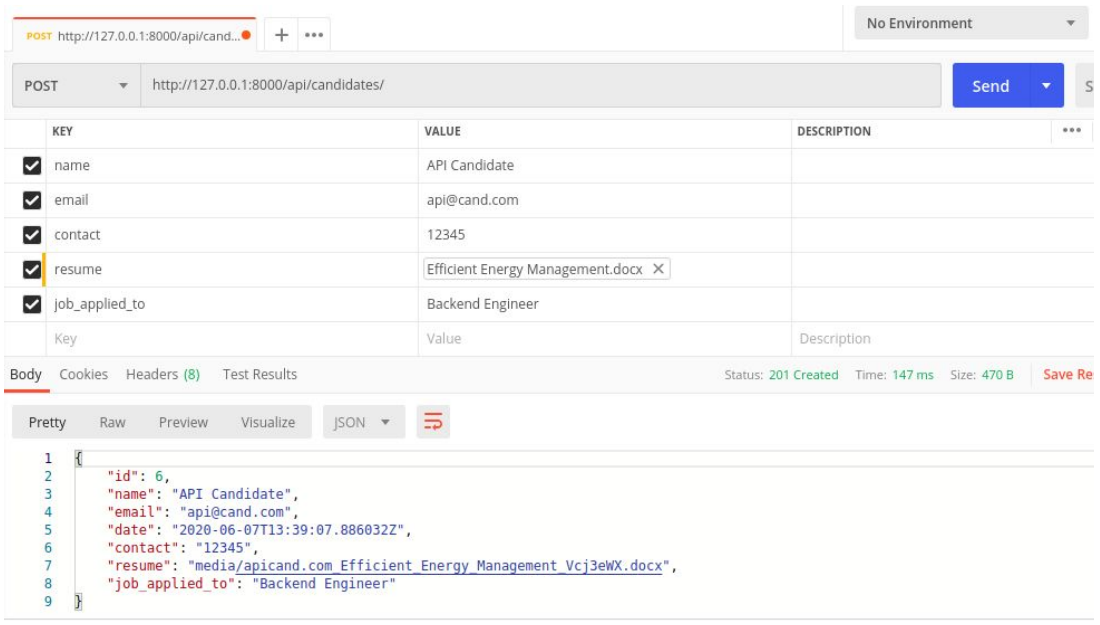
Params Authorization Headers (10) Body Pre-request Script Tests Settings

Headers 8 hidden

	KEY	VALUE
☐	Content-Type	application/json
✓	Authorization	Token dbd633f5ec54b2f1433336a9c696994fb8091630

6. Click **Send**. You'll receive **HTTP Status Code 201 Created** as well as a JSON Response containing the newly created Candidate.

End Result:


POST http://127.0.0.1:8000/api/cand...

No Environment

POST http://127.0.0.1:8000/api/candidates/ Send

	KEY	VALUE	DESCRIPTION
✓	name	API Candidate	
✓	email	api@cand.com	
✓	contact	12345	
✓	resume	Efficient Energy Management.docx	
✓	job_applied_to	Backend Engineer	
	Key	Value	Description

Body Cookies Headers (8) Test Results Status: 201 Created Time: 147 ms Size: 470 B Save Re

Pretty Raw Preview Visualize JSON

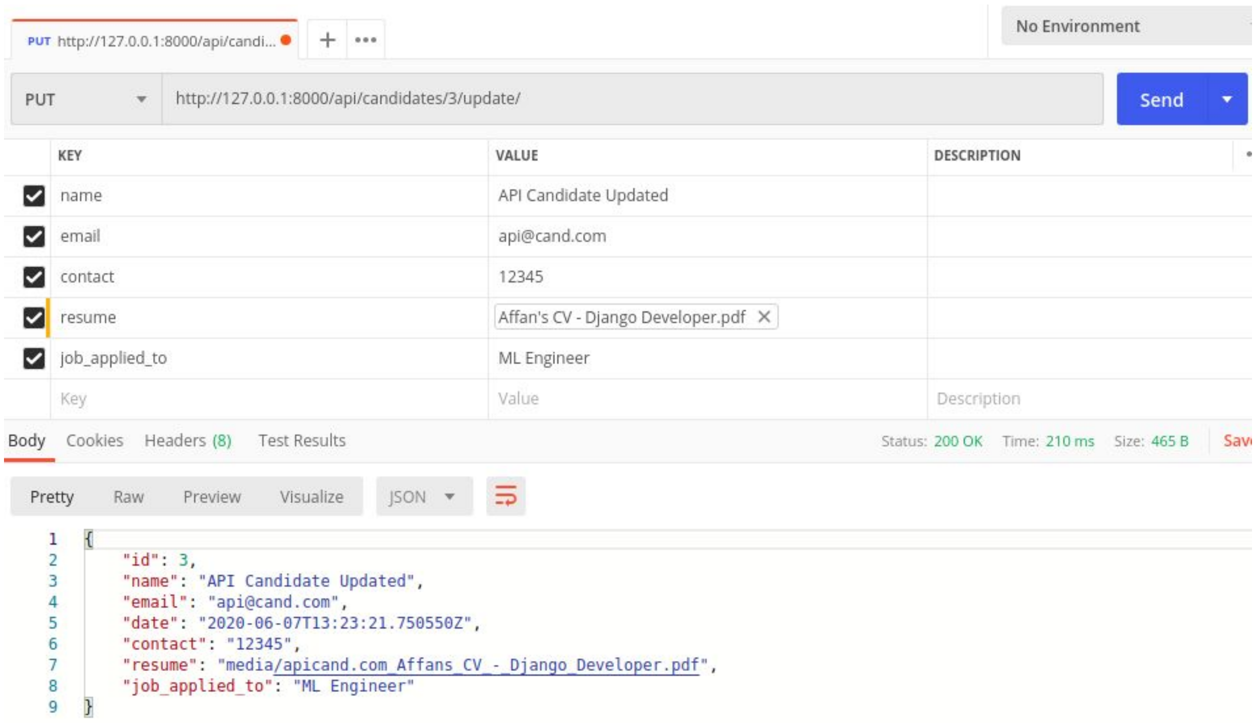
```
{
    "id": 6,
    "name": "API Candidate",
    "email": "api@cand.com",
    "date": "2020-06-07T13:39:07.886032Z",
    "contact": "12345",
    "resume": "media/apicand.com_Efficient_Energy_Management_Vcj3eWX.docx",
    "job_applied_to": "Backend Engineer"
}
```

Details To Update a Candidate

Use exactly the same steps as above (**Details To Add New Candidate**) except with a new endpoint and request type as follows. You'll receive HTTP Status Code 200 OK on successful operation:

Endpoint → http://127.0.0.1:8000/api/candidates/<id of candidate>/update/
Request type → PUT

End Result:



PUT http://127.0.0.1:8000/api/candidates/3/update/

KEY	VALUE	DESCRIPTION
name	API Candidate Updated	
email	api@cand.com	
contact	12345	
resume	Affan's CV - Django Developer.pdf	
job_applied_to	ML Engineer	

Status: 200 OK Time: 210 ms Size: 465 B

```
{
    "id": 3,
    "name": "API Candidate Updated",
    "email": "api@cand.com",
    "date": "2020-06-07T13:23:21.750550Z",
    "contact": "12345",
    "resume": "media/apicand.com_Affans_CV_-_Django_Developer.pdf",
    "job_applied_to": "ML Engineer"
}
```

Details To Delete a Candidate

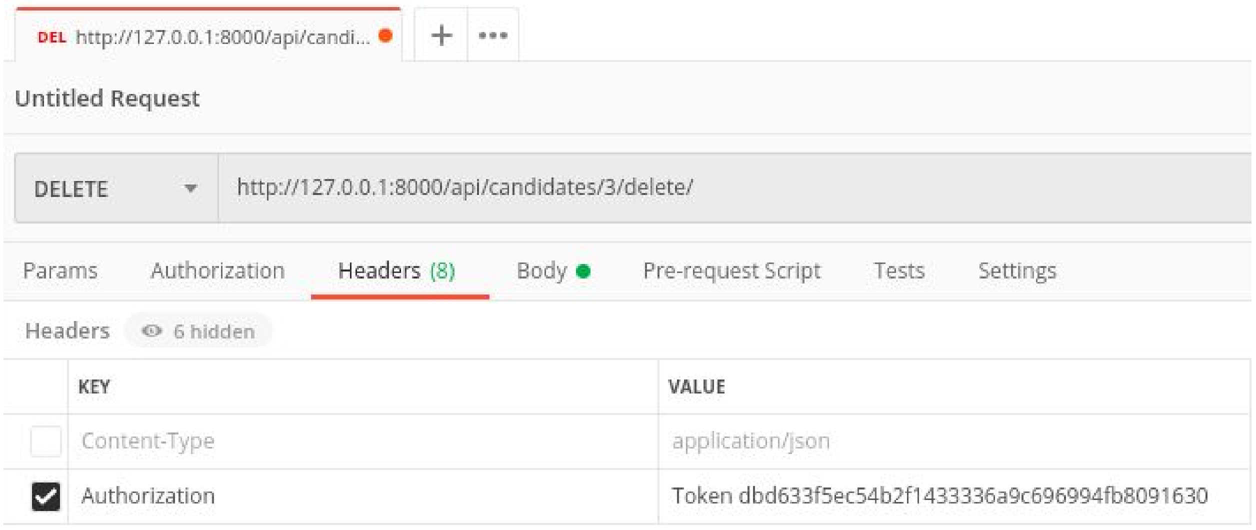
Use the following information to delete a candidate:

Endpoint → http://127.0.0.1:8000/api/candidates/<id of candidate>/delete/
Request type → DELETE

Steps to be done:

1. Click on **Body**.
2. **Uncheck all Key / Value** pairs you were sending previously. This process only requires authentication and no other parameters.
3. Now click on **Headers**.
4. Add the following Key / Value pair as follows:

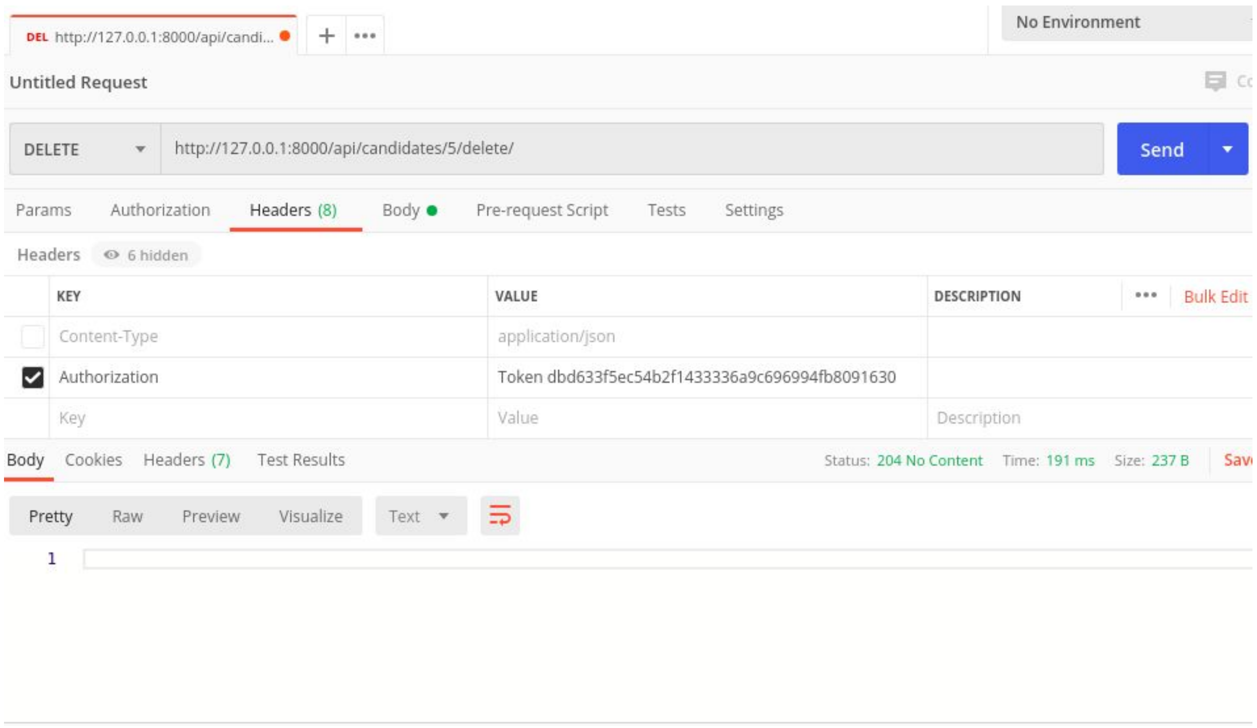
KEY	**VALUE**
Authorization	**Token** <auth_token obtained from signup> **NOTE:** The format contains the keyword **Token** then a **single space** and then the auth_token itself obtained when you signed up.



	KEY	VALUE
☐	Content-Type	application/json
✓	Authorization	Token dbd633f5ec54b2f1433336a9c696994fb8091630

5. Click **Send**. You'll receive **HTTP Status Code 204 No Content** on successful operation.

End Result:


DEL http://127.0.0.1:8000/api/candi...

No Environment

Untitled Request

DELETE http://127.0.0.1:8000/api/candidates/5/delete/

Headers (8)

	KEY	VALUE	DESCRIPTION
	Content-Type	application/json	
✓	Authorization	Token dbd633f5ec54b2f1433336a9c696994fb8091630	
	Key	Value	Description

Status: 204 No Content Time: 191 ms Size: 237 B

Details To Patch A Candidate

Use the following information to Patch a candidate:

Endpoint → http://127.0.0.1:8000/api/candidates/<id of candidate>/patch/
Request type → PATCH

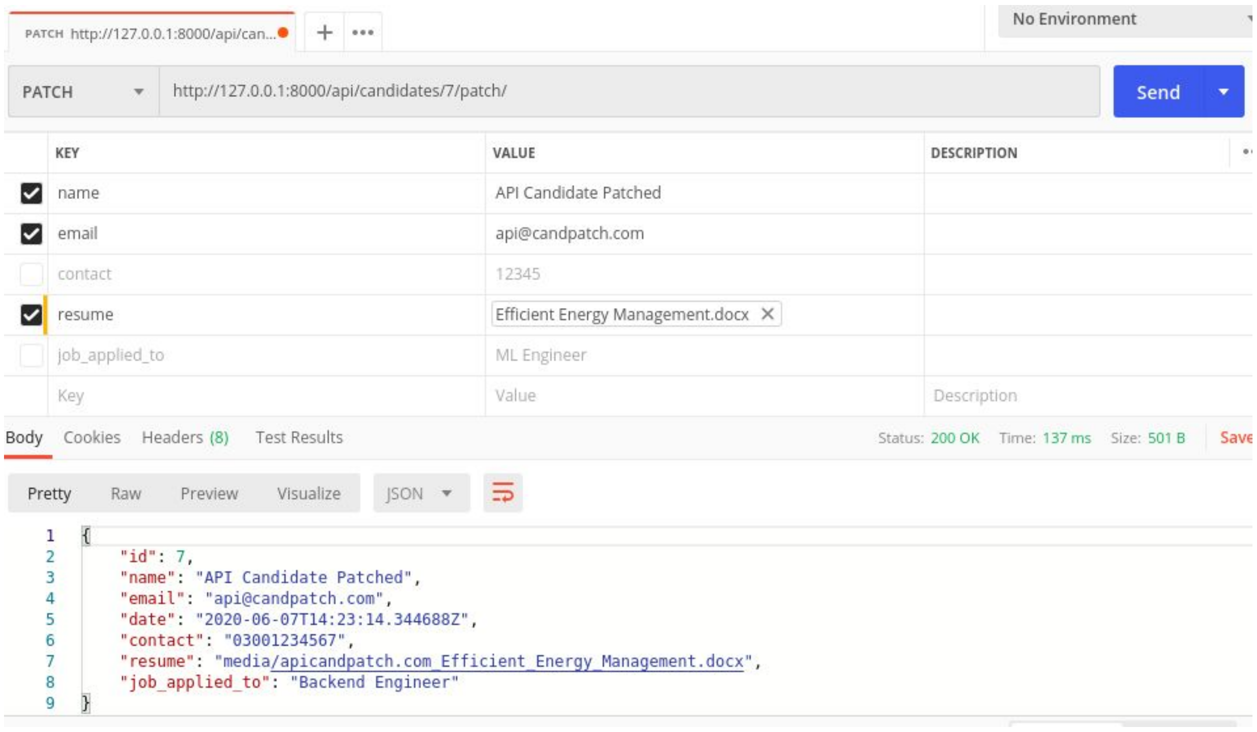
Steps to be done:

1. Click on **Body**.
2. Check the **form-data** radio button.
3. Add Key / Value pairs that you want to PATCH.
4. Now click on **Headers**.
5. Add key / value pair as follows:

KEY	VALUE
Authorization	**Token** <auth_token obtained from signup> **NOTE:** The format contains the keyword **Token** then a **single space** and then the auth_token itself obtained when you signed up.

6. Now click **Send**. You'll receive **HTTP Status Code 200 OK** on successful operation as well as a JSON response containing the PATCHed candidate.

End Result:



PATCH http://127.0.0.1:8000/api/candidates/7/patch/

	KEY	VALUE	DESCRIPTION
✓	name	API Candidate Patched	
✓	email	api@candpatch.com	
	contact	12345	
✓	resume	Efficient Energy Management.docx	
	job_applied_to	ML Engineer	
	Key	Value	Description

Status: 200 OK Time: 137 ms Size: 501 B

```
{
    "id": 7,
    "name": "API Candidate Patched",
    "email": "api@candpatch.com",
    "date": "2020-06-07T14:23:14.344688Z",
    "contact": "03001234567",
    "resume": "media/apicandpatch.com_Efficient_Energy_Management.docx",
    "job_applied_to": "Backend Engineer"
}
```