

REST API for an Electronic Voting System

Richard Nana Kweenu Quayson

29962024

Department of Computer Science and Information Systems, Ashesi University

CS 341_B: Lab 4 Build and run a REST API

Dennis Owusu

March 27 2023

Link to repository:

https://github.com/Richard-Quayson/Richard_Quayson_REST_API_LAB_4

Question 1:

Functionalities:

1. Register a student as a voter.
2. De-register a student as a voter (say when they leave campus).
3. Update a registered voter's information.
4. Retrieve a registered voter's information.
5. Create an election.
6. Retrieve an election.
7. Delete an election.
8. Vote in an election.

Design Decisions

Voter attributes:

- Student ID
- Firstname
- Lastname
- Email
- Year Group (not needed since it can be extracted from student's id)
- Is_registered (whether or not a user has been de-registered)

Election attributes:

- Election code
- Election name
- Election date
- Election period (measured in hours)
- Candidates
- Description

Register a voter:

Request: POST → URL: .../voters/register_voter/

{“student_ID”: 22332024, “firstname”: “Raymond”, “lastname”: “Negrefos”, “email”:
“raymond.negrefos@ashesi.edu.gh”}

Response: STATUS → 200 (OK)

{“student_ID”: 22332024, “firstname”: “Raymond”, “lastname”: “Negrefos”, “email”:
“raymond.negrefos@ashesi.edu.gh”}

Deregister a student as a voter:

Request: PATCH → URL: .../voters/de-register/22332024/ (student_id)

Or URL: .../voters/de-register/2024/ (year_group)

Response: STATUS: → 200 OK

{“message”: “Student has been successfully de-registered.”, “student_ID”: 22332024,
“firstname”: “Raymond”, “lastname”: “Negrefos”, “is_registered”: False }

Update registered voter’s information:

Request: PATCH → URL: .../update-voter/22332024/

Response: STATUS: → 200 (OK)

{“message”: “Student info has been successfully updated.”, “student_ID”: 22332024,
“firstname”: “Raymond”, “lastname”: “Negrefos”, “email”:
“raymond.negrefos@ashesi.edu.gh”}

Retrieve a registered voter:

Request: GET → URL: .../retrieve-voter/22332024/

Response: STATUS: → 200 (OK)

{“student_ID”: 22332024, “firstname”: “Raymond”, “lastname”: “Negrefos”, “email”:
“raymond.negrefos@ashesi.edu.gh”}

Create an election:

Request: POST → URL: .../create-election/

{“election_code”: “2023ASC”, “election_name”: “2023ASCElection”, “election_startdate”:
“2022-03-25”, “election_period”: “72”, “candidates”: [22332024, 29942024, 19872025],
“election_description”: “ASC and JEC election for the 2023/2024 ASC heads.”}

Response: STATUS → 200 (OK)

{“election_code”: “2023ASC”, “election_name”: “2023ASCElection”, “election_startdate”:
“2022-03-25”, “election_period”: “72”, “candidates”: [22332024, 29942024, 19872025],
“election_description”: “ASC and JEC election for the 2023/2024 ASC heads.”}

Retrieve an election:

Request: GET → URL: .../election info/?<any attribute of election>

Response: STATUS → 200 (OK)

```
{“election_code”: “2023ASC”, “election_name”: “2023ASCElection”, “election_startdate”:  
“2022-03-25”, “election_period”: “72”, “candidates”: [22332024, 29942024, 19872025],  
“election_description”: “ASC and JEC election for the 2023/2024 ASC heads.”}
```

Delete an election:

Request: DELETE ➔ URL: .../delete-election/<election_code>/

Response: STATUS: ➔ 204 (No content)

```
{“message”: “Election has been successfully deleted.”, “election_code”: “2023ASC”,  
“election_name”: “2023ASCElection”}
```

Vote in an election:

Request: POST ➔ URL: .../vote/

```
{“election_code”: “2023ASC”, “candidate_ID”: “22332024”}
```

Response: STATUS ➔ 200 (OK)

```
{“message”: “You have successfully voted for student with the following details”,  
“election_code”: “2023ASC”, “student_ID”: 22332024, “firstname”: “Raymond”,  
“lastname”: “Negrefos”}
```

TESTING ENDPOINTS

1. Register a voter.

When no data is provided, the API throws a bad request with the appropriate message of missing voter information.

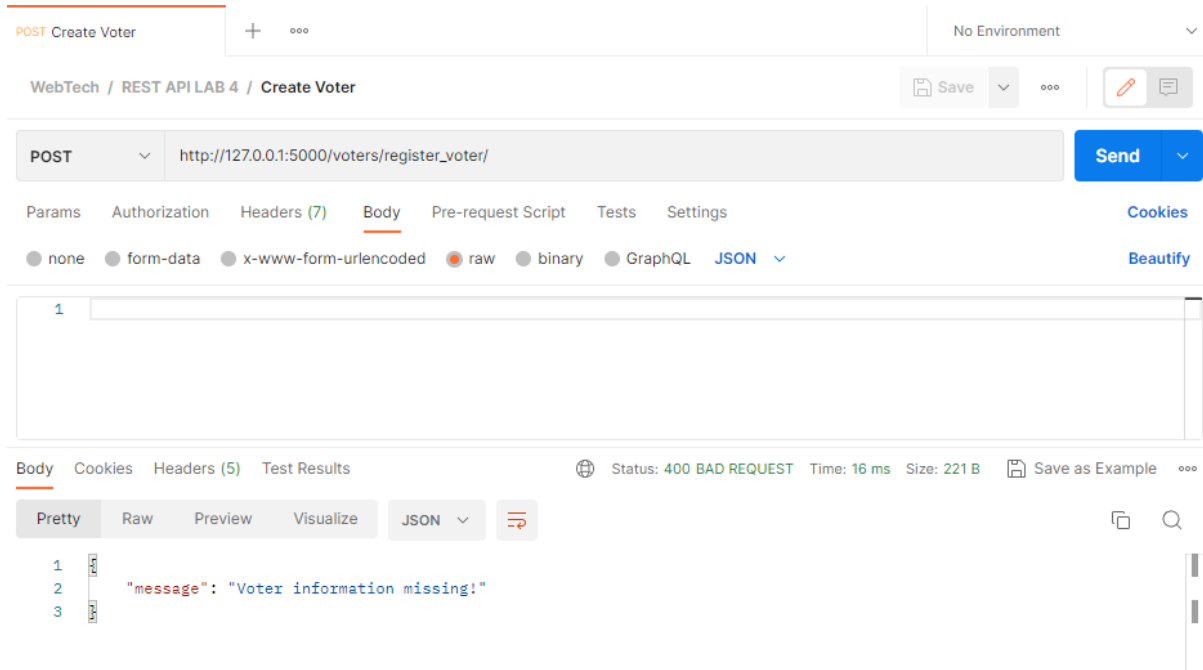


Figure 1 shows a POST request for registering a user when no data is provided.

When incomplete data (data missing some attributes) is provided, the API throws a bad request with the appropriate message of attribute(s) required.

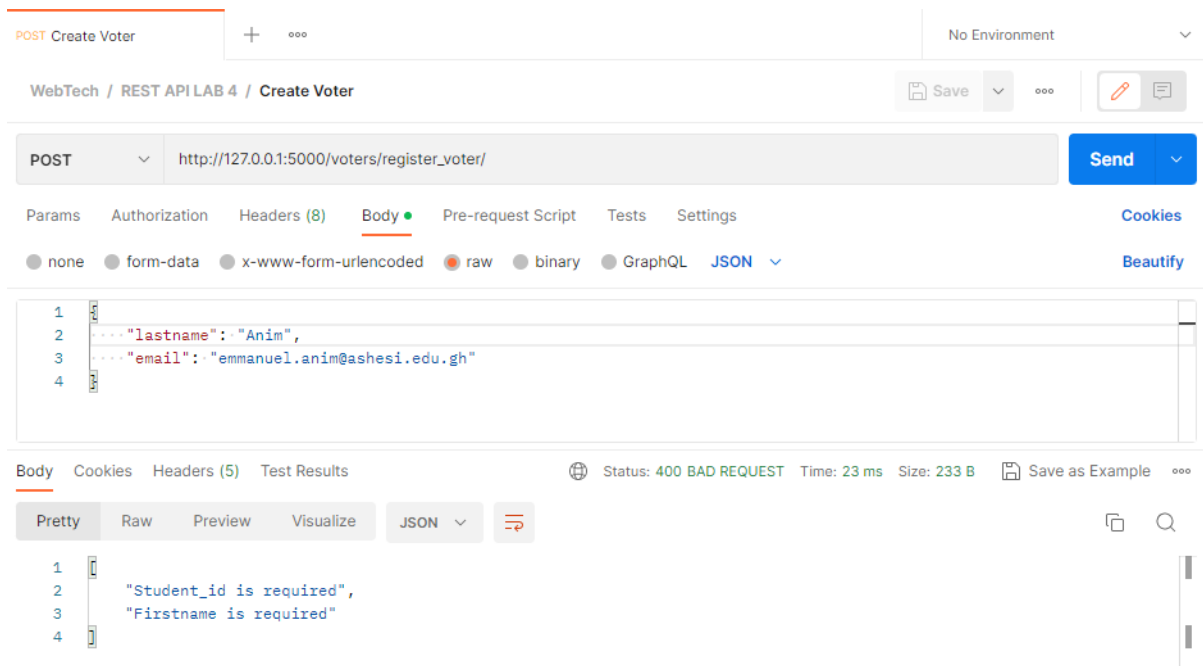


Figure 2 shows a POST request for registering a user when insufficient data is provided.

When an invalid student id is provided, the API throws a bad request with the appropriate message of student id not valid. An id is valid if it is numeric and consists of 8 characters.

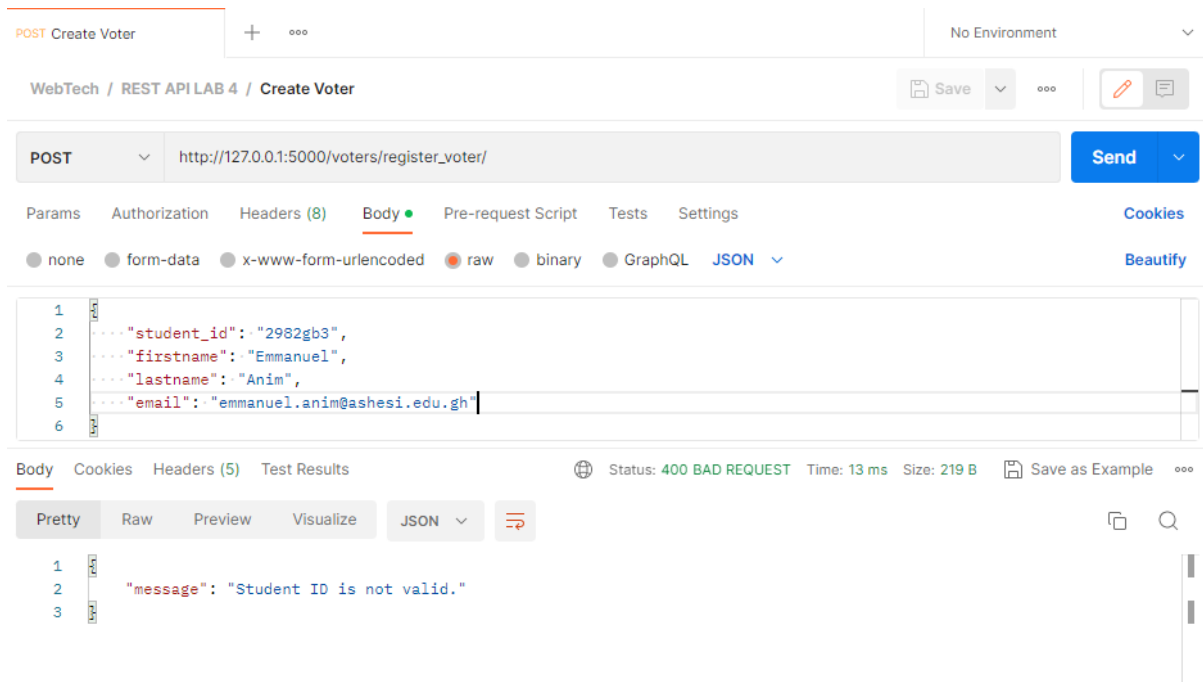


Figure 3 shows a POST request for registering a user when an invalid student id is provided.

When an invalid year group is provided, the API throws a bad request with the appropriate message of year group is invalid. Note: The year group is the last four digit of an Ashesi student id.

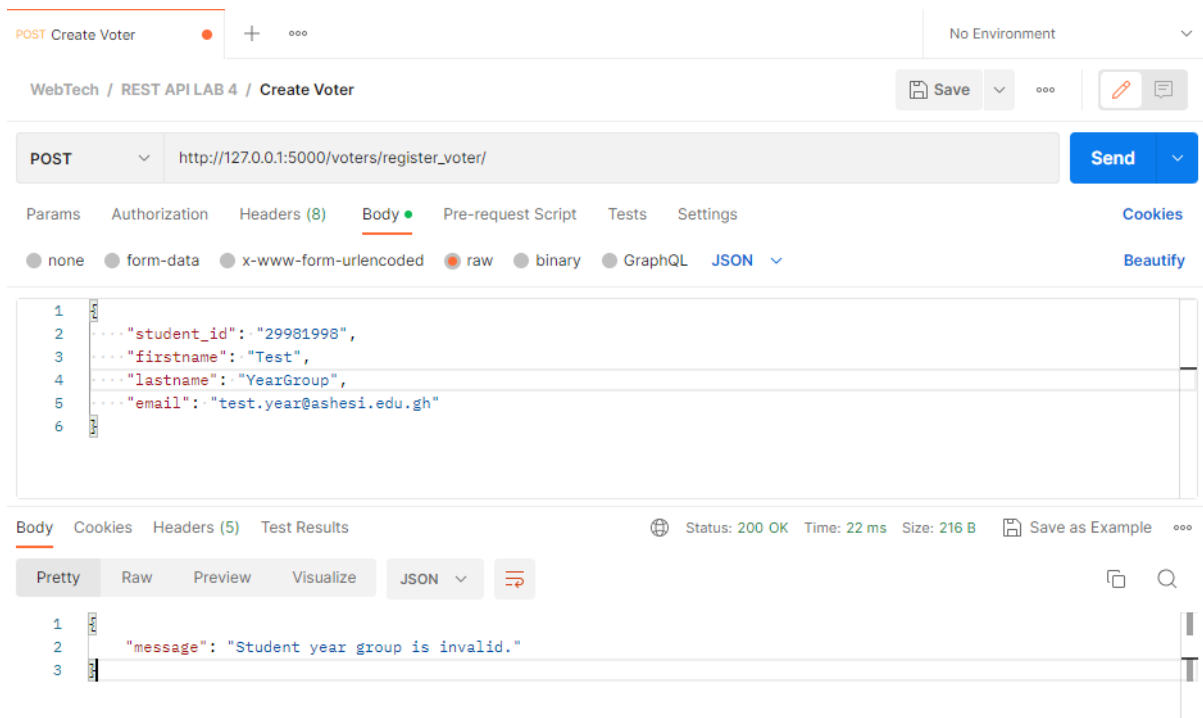


Figure 4 shows a POST request for registering a user when an invalid student id is provided.

When an invalid firstname or lastname is provided, the API throws a bad request with the appropriate message of firstname and lastname must be string.

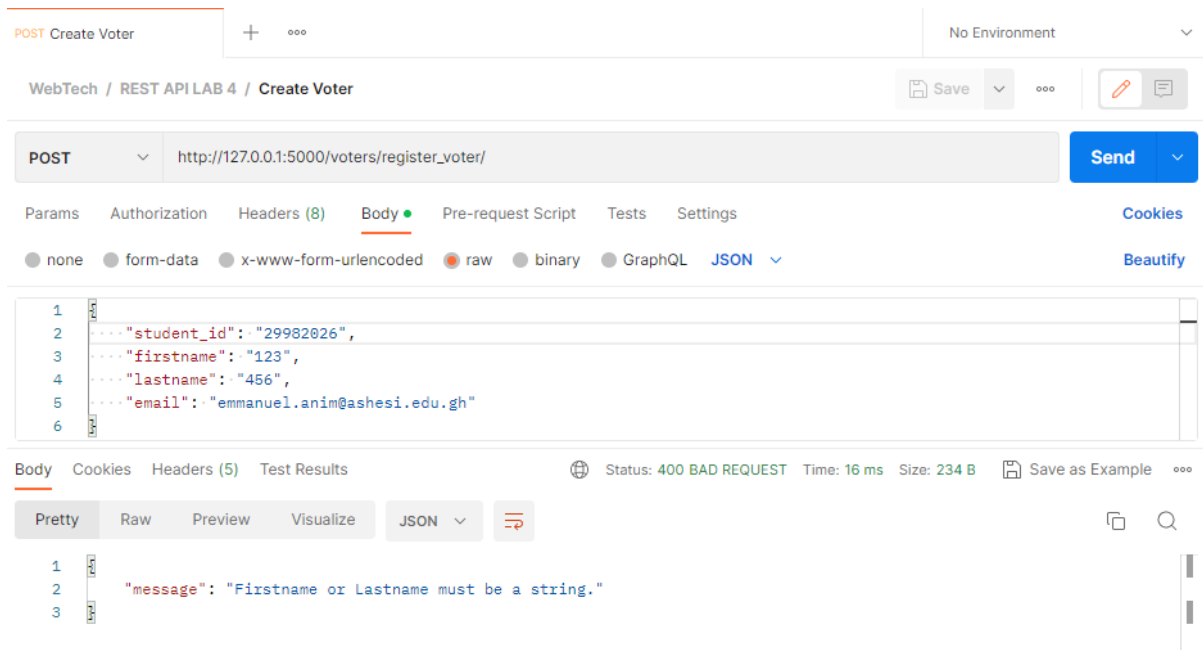


Figure 5 shows a POST request for registering a user when an invalid firstname or lastname is provided.

When an invalid email domain is identified, the API throws a bad request with the appropriate message of email must be a valid Ashesi email. **Note:** this validation doesn't include the username and as such any application using this API must perform regex validation on the email.

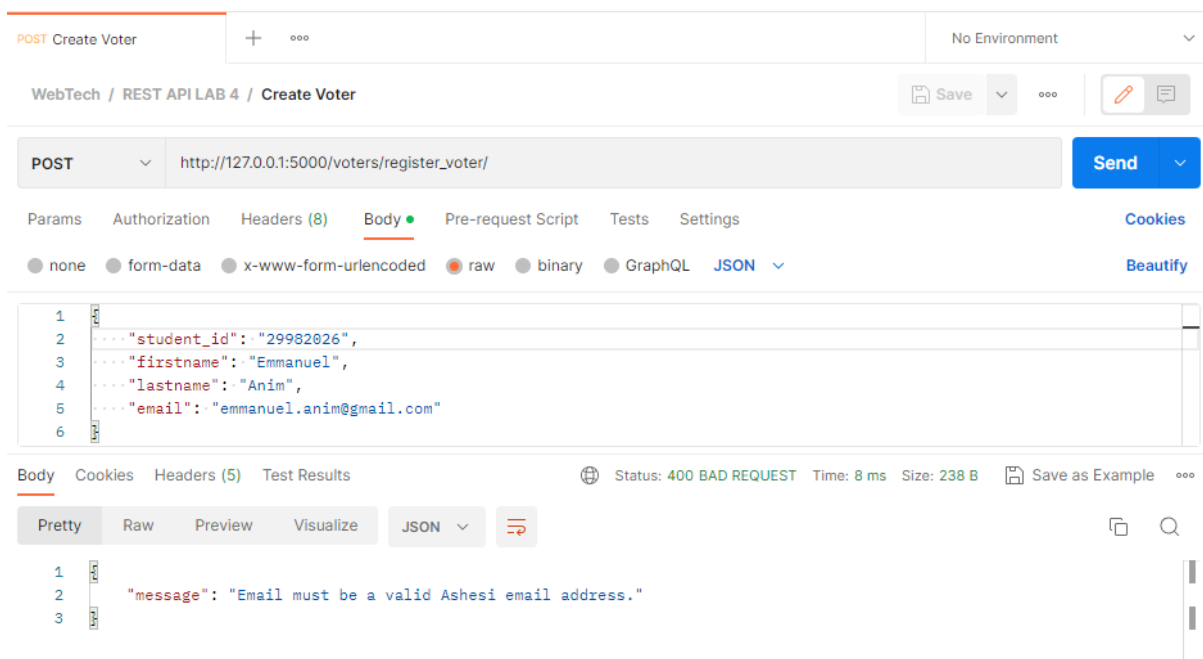


Figure 6 shows a POST request for registering a user when an invalid Ashesi email domain is provided.

Student id and email are unique fields and as such, when a provided student id or email already exists in the database, the API throws a bad request with the appropriate message of student id and email already exists.

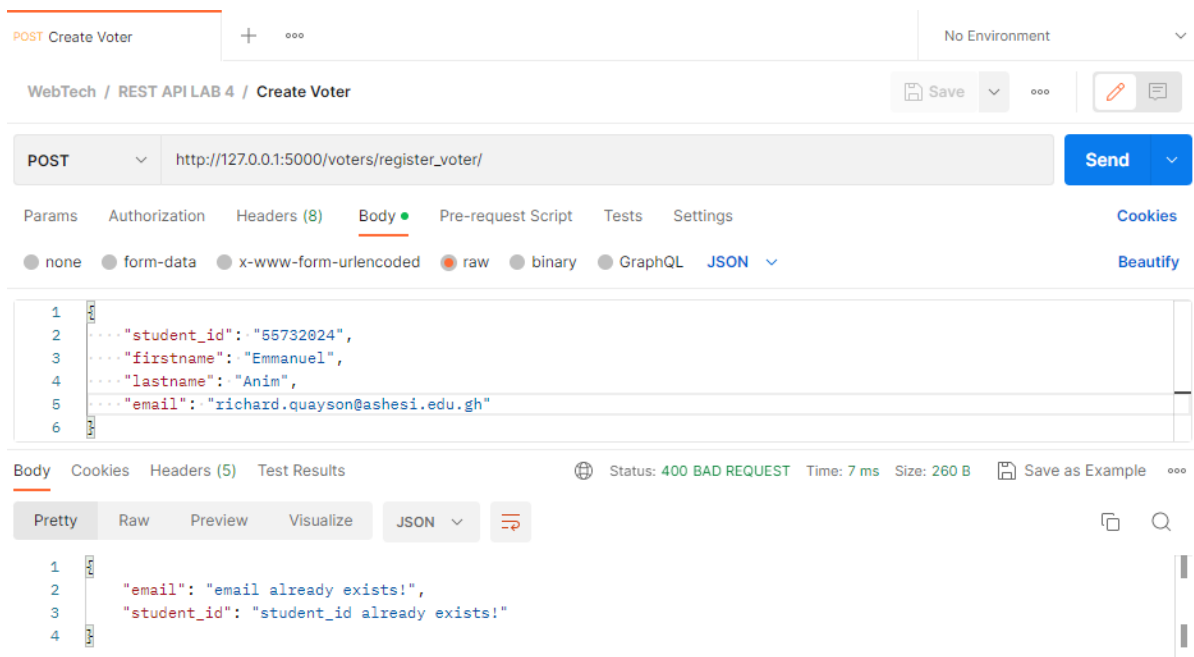


Figure 7 shows a POST request for registering a user when student id or email is not unique.

If all validation and unique constraints check are successful, the API returns a success and adds the new user to the list of voters with a `is_registered` attribute added and set to `True`. The `is_registered` attribute is used to check if a voter is registered.

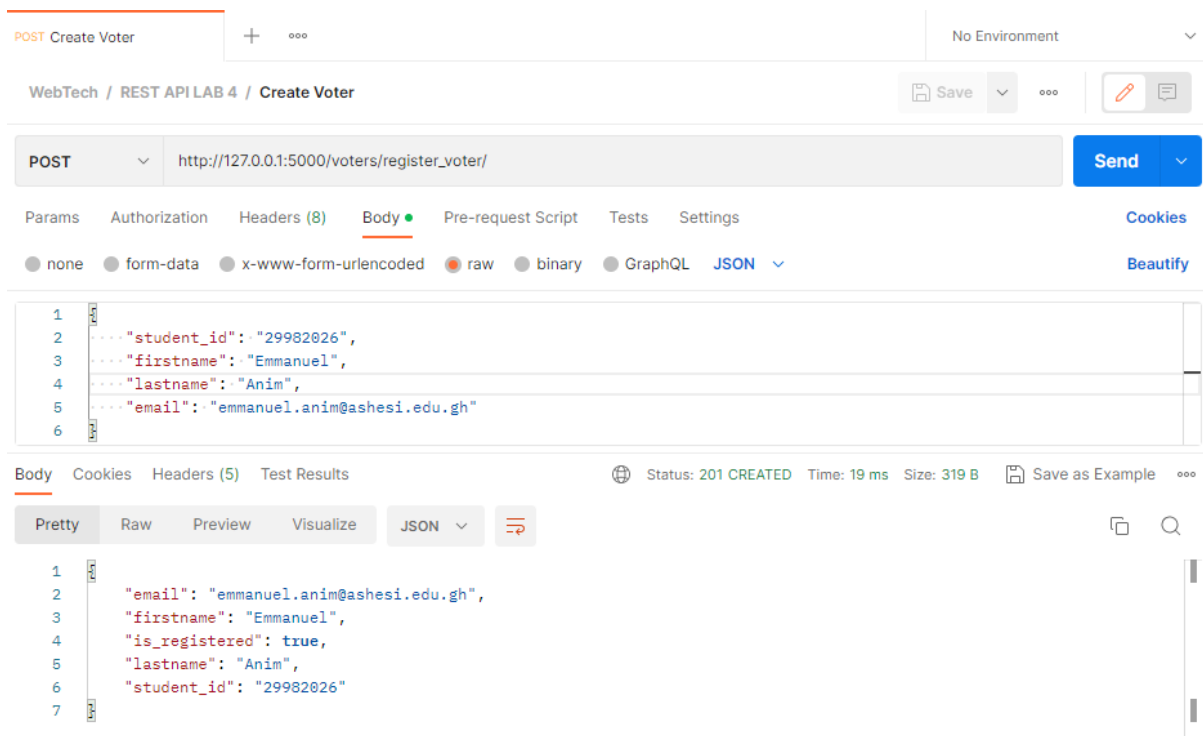


Figure 8 shows a POST request for registering a voter when all data is valid.

2. Deregister a student as a voter.

When a valid year group is parsed in the URL, the API returns either a not found when no user with the specified year group exists or a success when the voters with the year group exists.

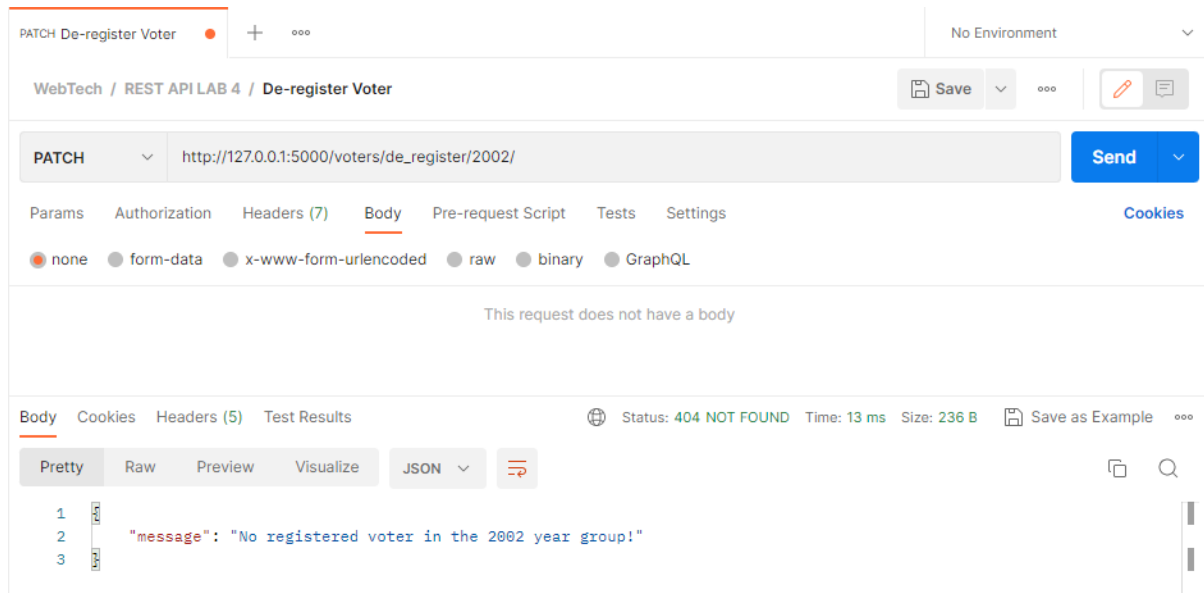


Figure 9 shows a PATCH request for deregistering a voter when no registered voter exists in the year group provided (2002).

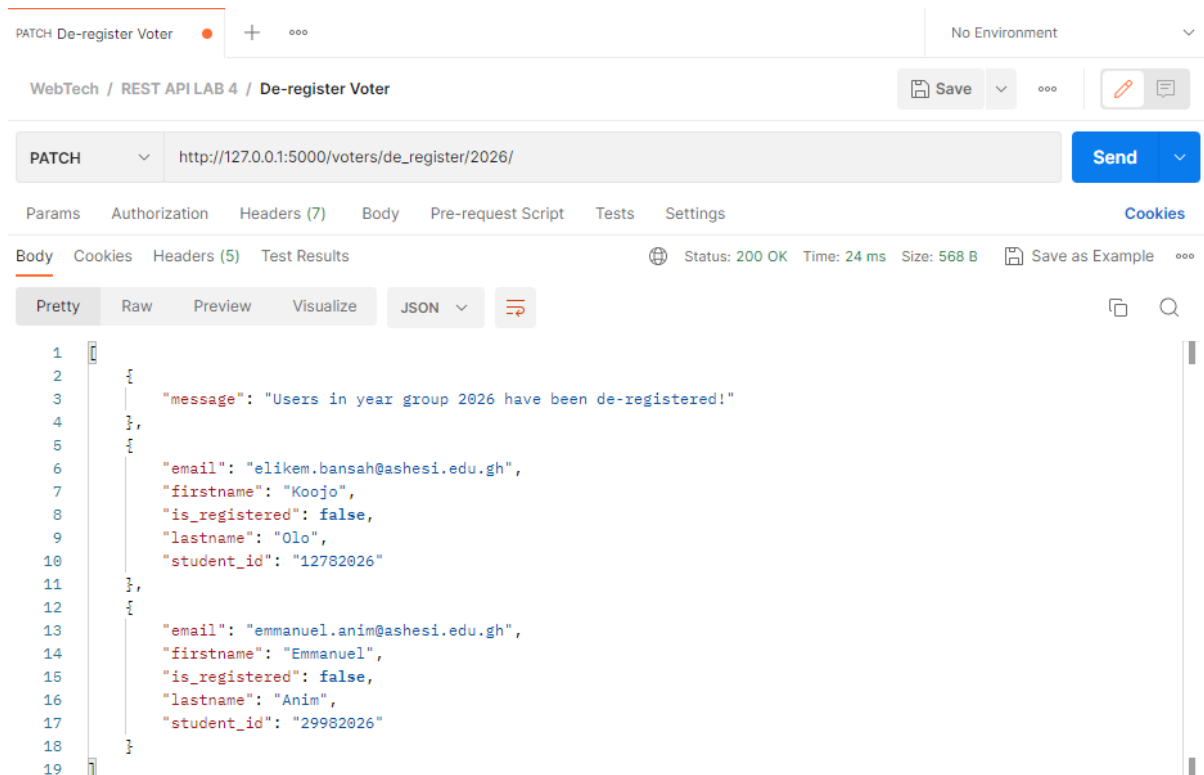


Figure 10 shows a PATCH request for deregistering voters when registered voters exists in the year group provided (2026).

Note: If the value passed into the endpoint for deregistering a student fails the year group constraint, it is considered as a student id.

When an invalid student id is parsed in the URL, the API returns a bad request with the appropriate response of invalid student id.

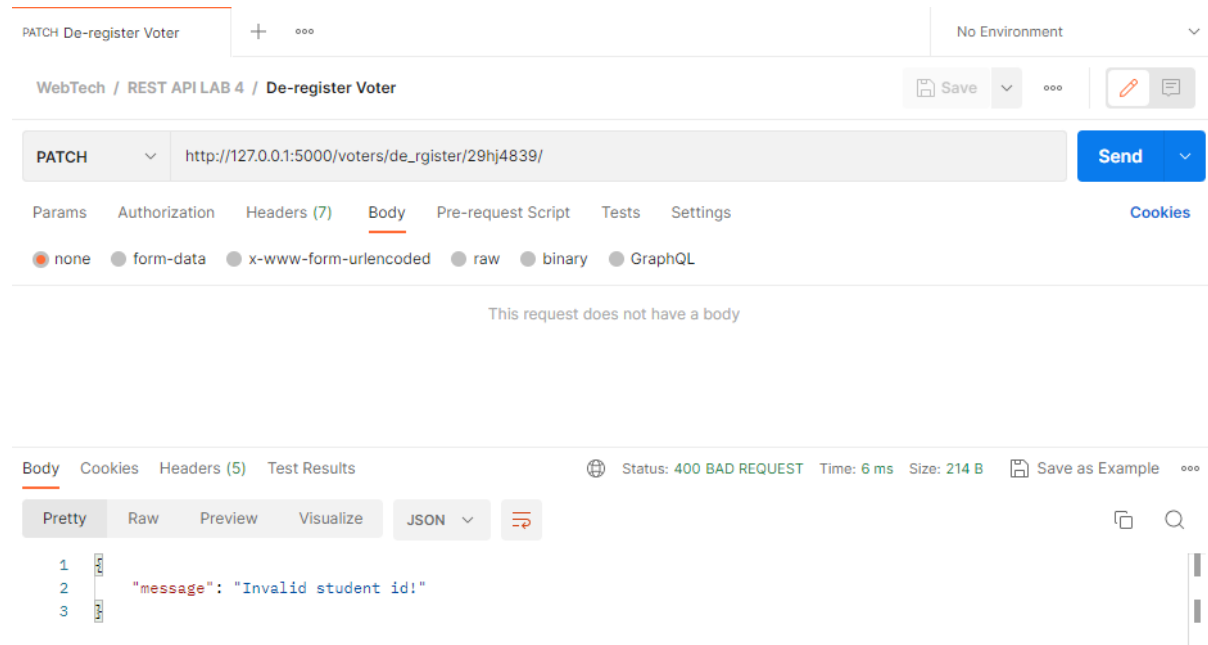


Figure 11 shows a PATCH request for de-registering a voter when student id is invalid.

When a voter with specified id is not found, the API returns a not found with the appropriate message that the student with specified id does not exist.

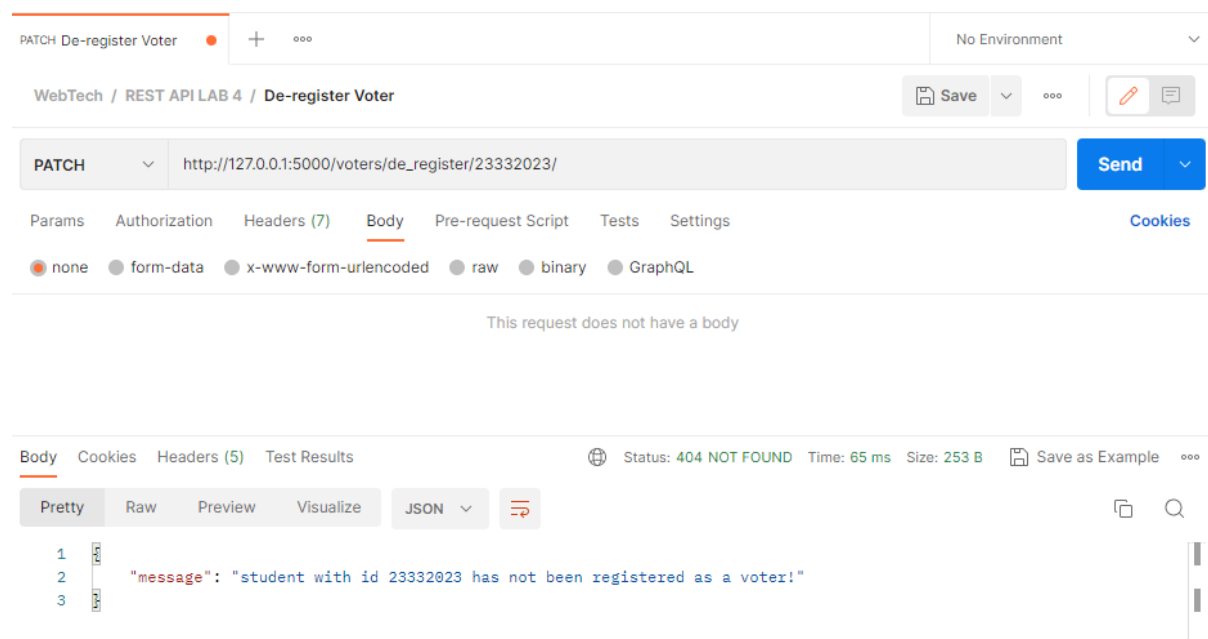


Figure 12 shows a PATCH request for de-registering a voter when student id does not exist in voters file.

If a voter with the specified student id exists, the API updates the value of `is_registered` (set to false) for the specified voter and returns the voter's updated information.

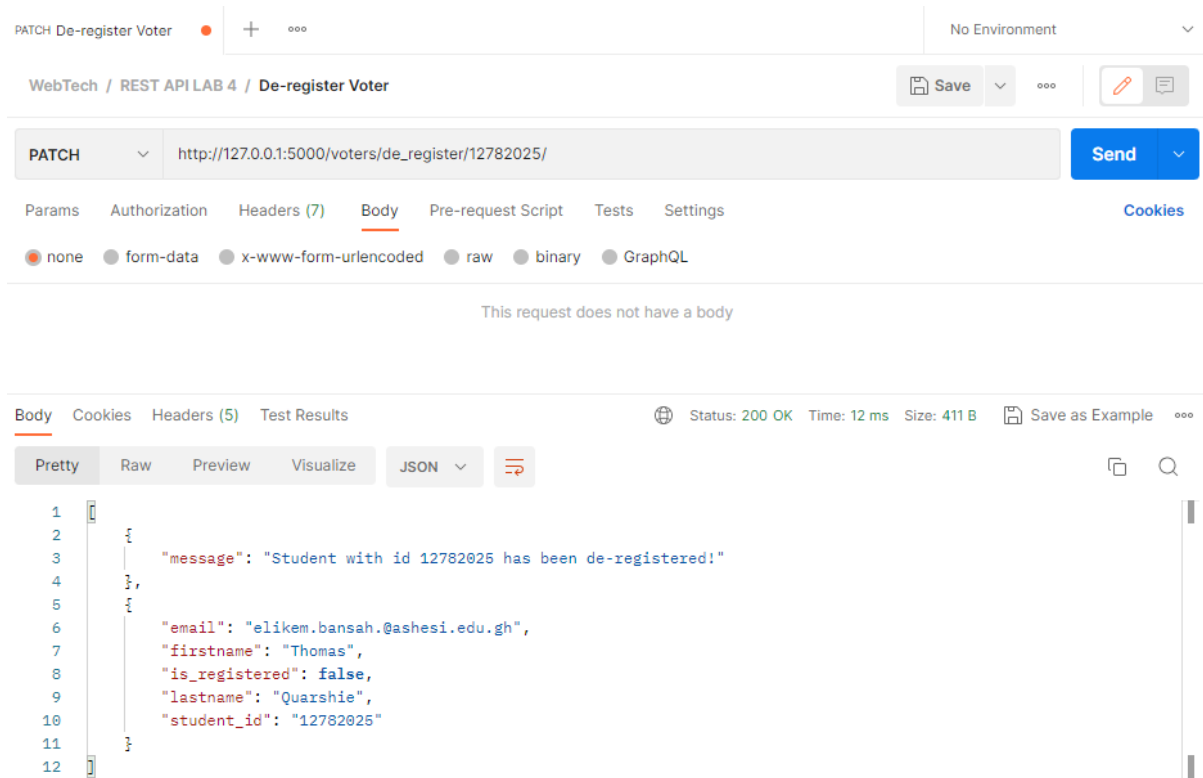


Figure 13 shows a PATCH request for de-registering a voter when the student id is valid, and the student has been registered as a voter.

3. Update a registered voter's information.

Note: All previous constraints are put in place. Not tested here to reduce the number of test results.

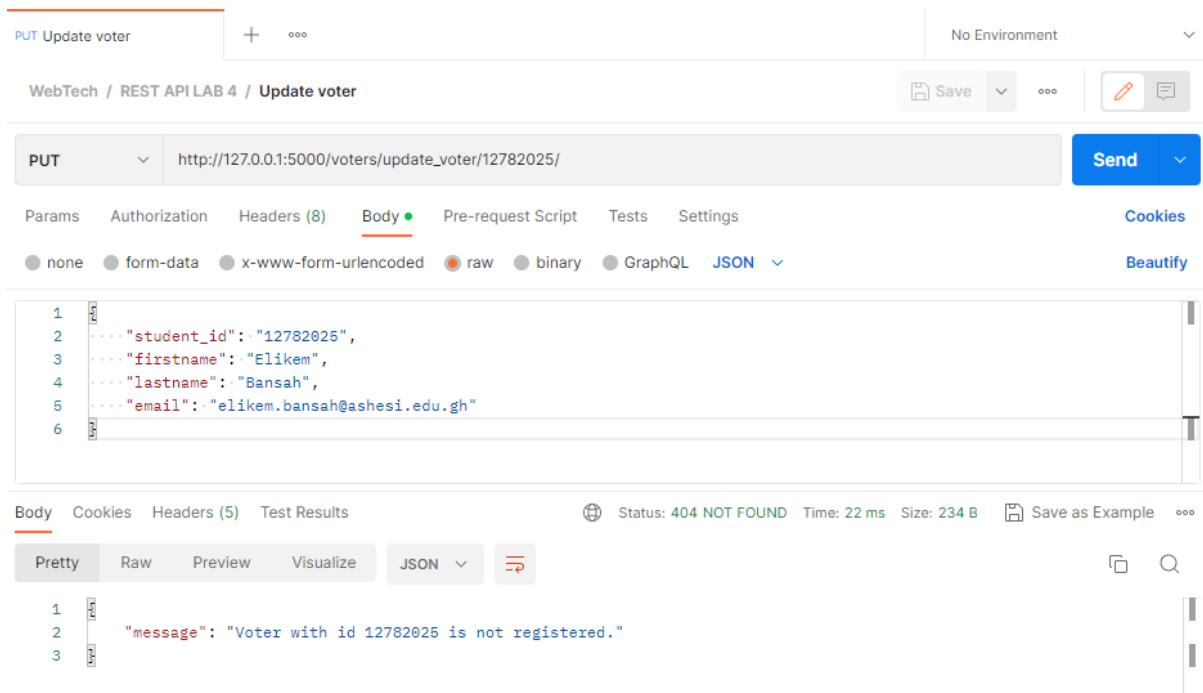
When the user is not registered, the API returns a bad request with the appropriate message of voter with specified id is not registered.

```

{
  "student_id": "12782025",
  "firstname": "Thomas",
  "lastname": "Quarshie",
  "email": "elikem.bansah@ashesi.edu.gh",
  "is_registered": false
},

```

Figure 14 shows the initial information for voter with student_id, 12782025.



PUT Update voter

WebTech / REST API LAB 4 / Update voter

PUT `http://127.0.0.1:5000/voters/update_voter/12782025/` **Send**

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "student_id": "12782025",
3   "firstname": "Elikem",
4   "lastname": "Bansah",
5   "email": "elikem.bansah@ashesi.edu.gh"
6 }
```

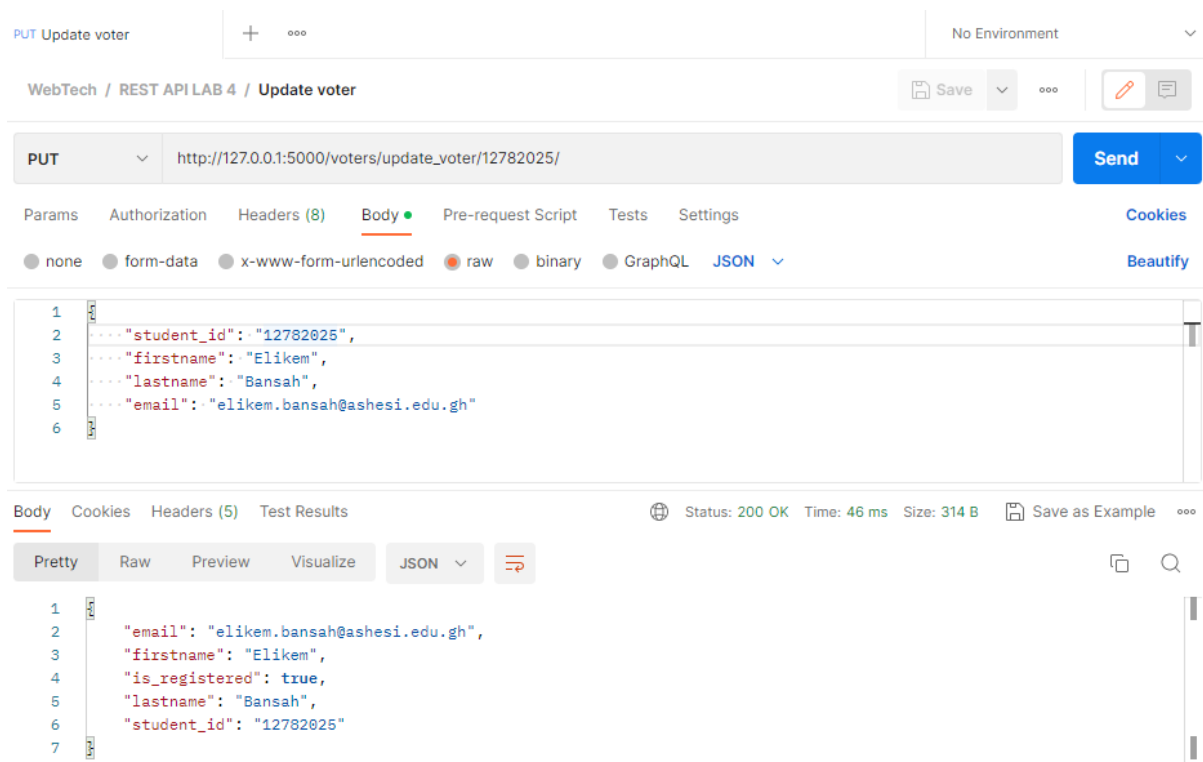
Body Cookies Headers (5) Test Results Status: 404 NOT FOUND Time: 22 ms Size: 234 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Voter with id 12782025 is not registered."
3 }
```

Figure 15 shows a PUT request for updating a voter when the voter with specified id is not registered as a voter.

If the `is_registered` is set to true for the user above and all constraints and validations are passed, the API updates all the attribute values of the voter with the specified id and returns the result.



PUT Update voter

WebTech / REST API LAB 4 / Update voter

PUT `http://127.0.0.1:5000/voters/update_voter/12782025/` **Send**

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "student_id": "12782025",
3   "firstname": "Elikem",
4   "lastname": "Bansah",
5   "email": "elikem.bansah@ashesi.edu.gh"
6 }
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 46 ms Size: 314 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "email": "elikem.bansah@ashesi.edu.gh",
3   "firstname": "Elikem",
4   "is_registered": true,
5   "lastname": "Bansah",
6   "student_id": "12782025"
7 }
```

Figure 16 shows a PUT request for updating a voter when the voter is registered and all constraints are met.

4. Retrieve registered voter(s).

If no filter is specified, the API returns all voters in the database.

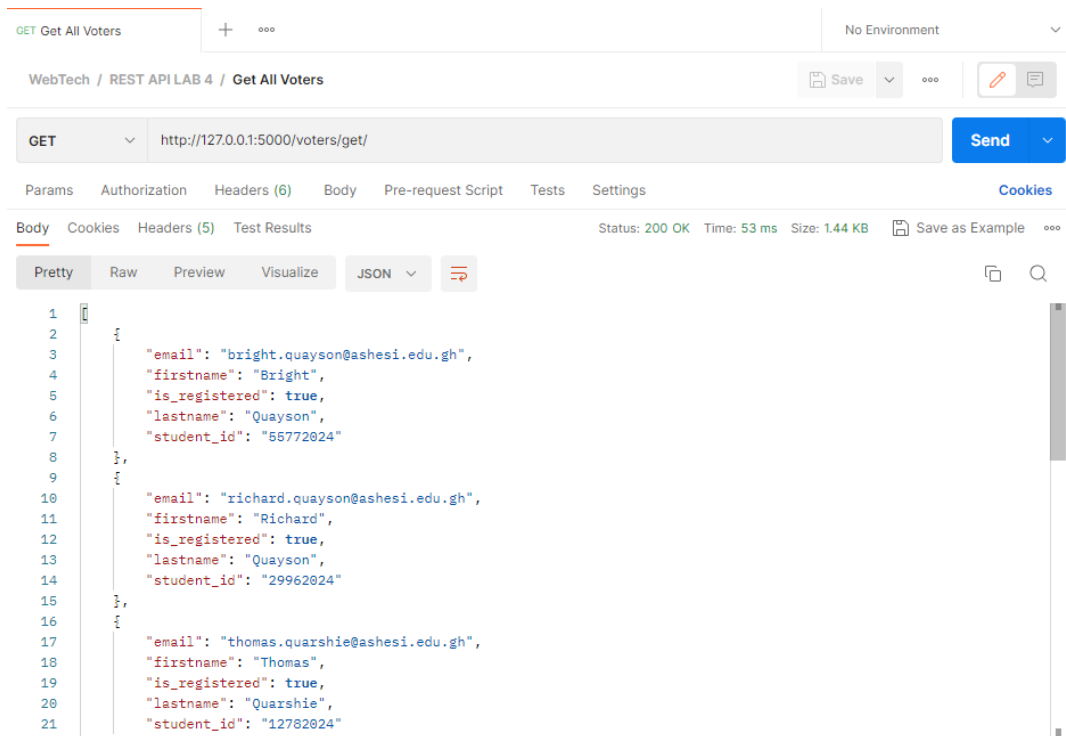


Figure 17 shows a GET request for retrieving voter information when no filter is specified.

If filters are applied, the API return the result the result if it exists and a not found with an appropriate message if no voter with the specified information exists.

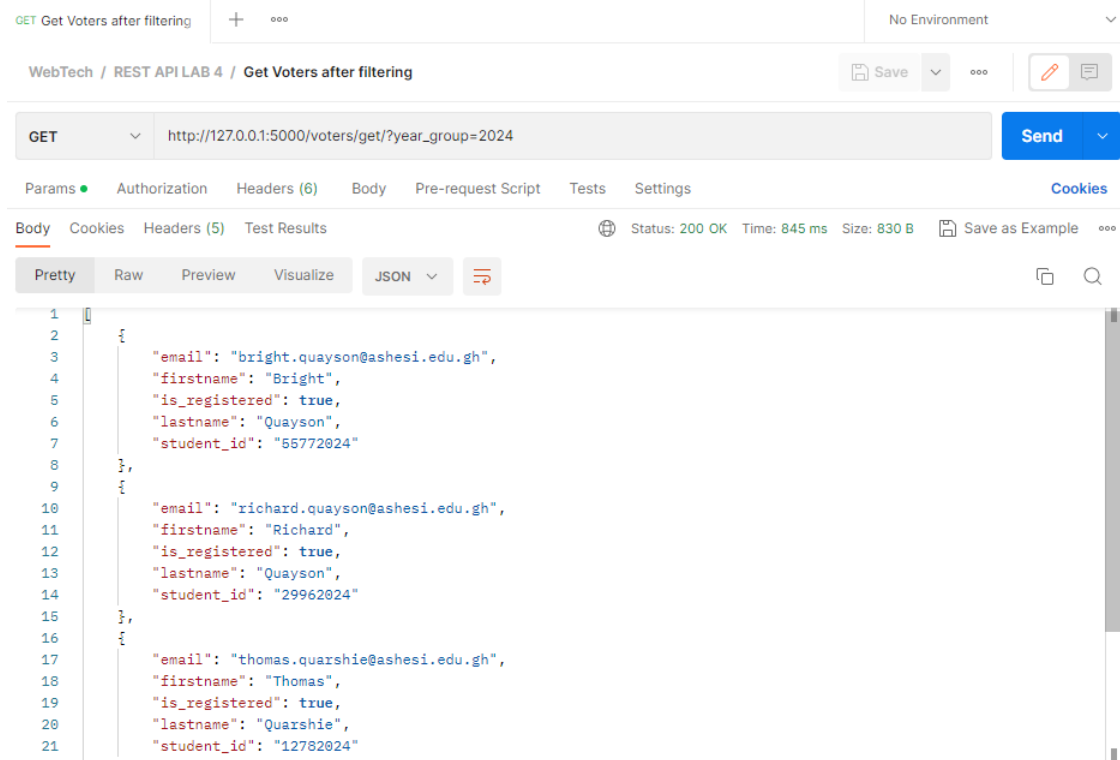


Figure 18 shows a GET request for retrieving voter information when a year group (2024) is specified.

The screenshot shows a REST client interface with the following details:

- Request:** GET `http://127.0.0.1:5000/voters/get/?year_group=2024&lastname=Quayson`
- Status:** 200 OK, Time: 10 ms, Size: 500 B
- Response Body (JSON):**

```
[{"email": "bright.quayson@ashesi.edu.gh", "firstname": "Bright", "is_registered": true, "lastname": "Quayson", "student_id": "55772024"}, {"email": "richard.quayson@ashesi.edu.gh", "firstname": "Richard", "is_registered": true, "lastname": "Quayson", "student_id": "29962024"}]
```

Figure 19 shows a GET request for retrieving voter information when a year group (2024) and lastname (Quayson) are specified.

The screenshot shows a REST client interface with the following details:

- Request:** GET `http://127.0.0.1:5000/voters/get/?year_group=2024&firstname=Morgan`
- Status:** 200 OK, Time: 19 ms, Size: 334 B
- Response Body (JSON):**

```
[{"email": "morgan.sarpong@ashesi.edu.gh", "firstname": "Morgan", "is_registered": true, "lastname": "Sarpong", "student_id": "55732024"}]
```

Figure 20 shows a GET request for retrieving voter information when a year group (2024) and firstname (Morgan) are specified.

The screenshot shows a REST client interface with the following details:

- Request Method:** GET
- URL:** `http://127.0.0.1:5000/voters/get/?email=thomas.quarshie@ashesi.edu.gh`
- Status:** 200 OK
- Time:** 9 ms
- Size:** 336 B
- Body (JSON):**

```
{  "email": "thomas.quarshie@ashesi.edu.gh",  "firstname": "Thomas",  "is_registered": true,  "lastname": "Quarshie",  "student_id": "12782024"}
```

Figure 21 shows a GET request for retrieving voter information when an email (Thomas.quarshie@ashesi.edu.gh) is specified.

The screenshot shows a REST client interface with the following details:

- Request Method:** GET
- URL:** `http://127.0.0.1:5000/voters/get/?student_id=29962024`
- Status:** 200 OK
- Time:** 7 ms
- Size:** 336 B
- Body (JSON):**

```
{  "email": "richard.quayson@ashesi.edu.gh",  "firstname": "Richard",  "is_registered": true,  "lastname": "Quayson",  "student_id": "29962024"}
```

Figure 22 shows a GET request for retrieving voter information when a student_id (29962024) is specified.

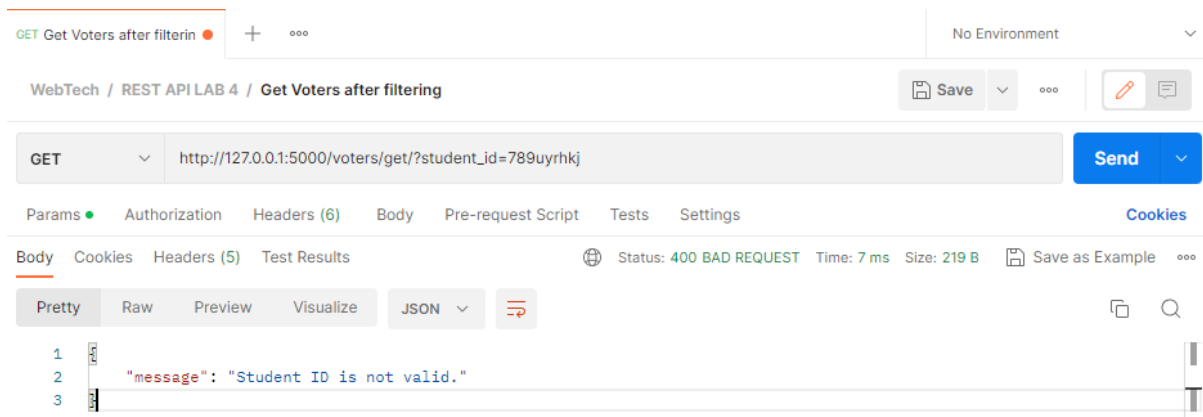


Figure 23 shows a GET request for retrieving voter information when an invalid `student_id` is specified.

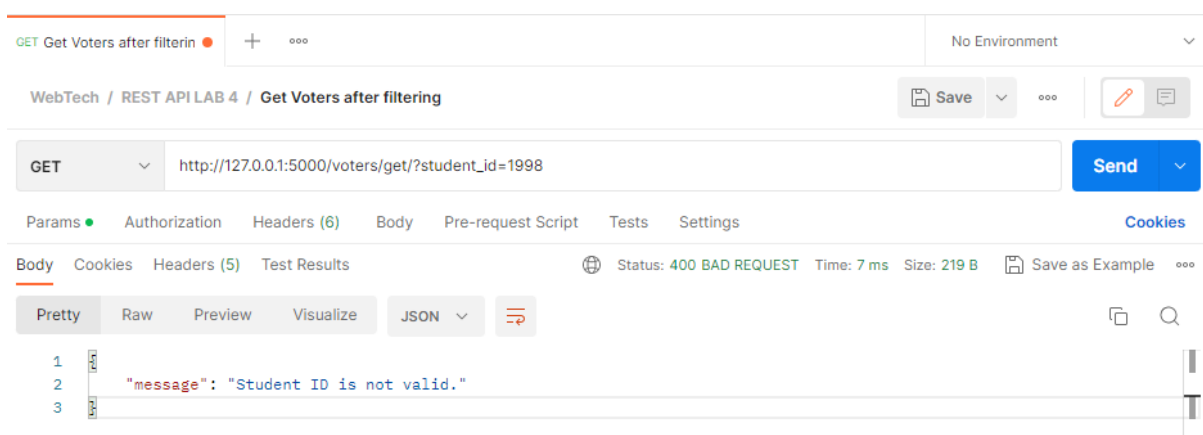


Figure 24 shows a GET request for retrieving voter information when an invalid `year_group` is specified.

Note: All other input validation checks used above is applied here too.

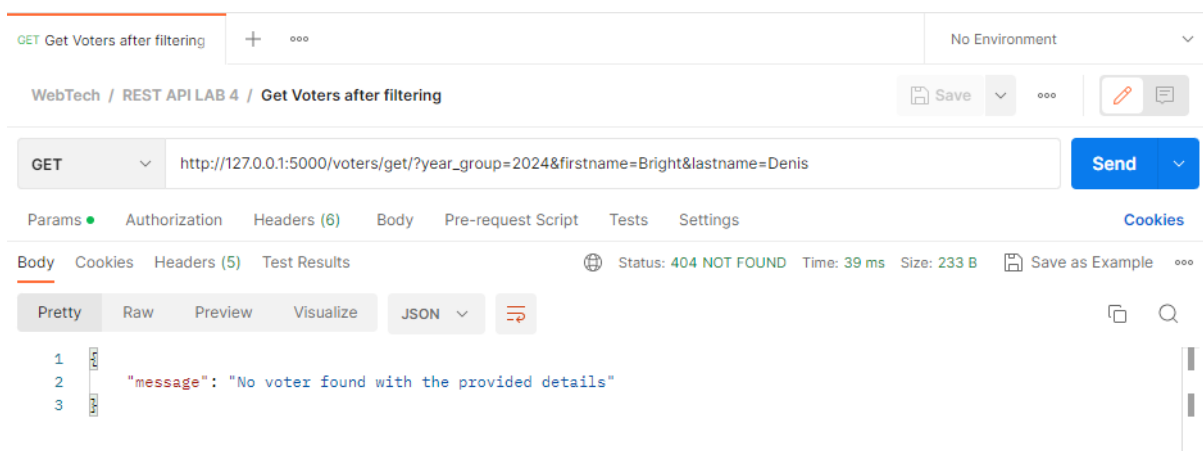


Figure 25 shows a GET request for retrieving voter information when details specified does not match any voter in the database.

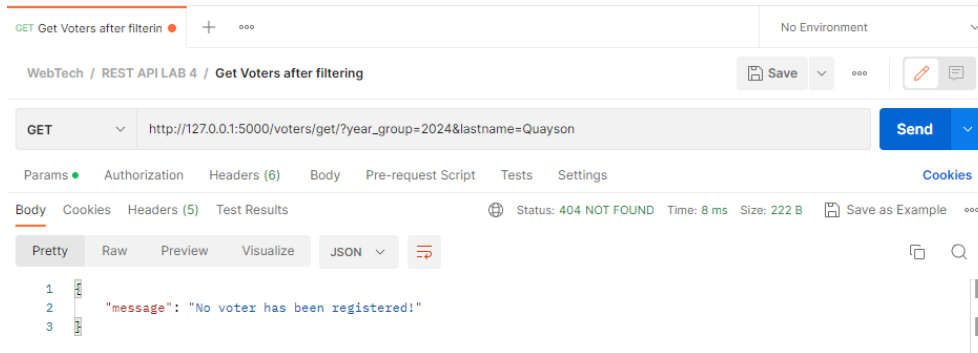


Figure 26 shows a GET request for retrieving voter information when there is not voter in the database.

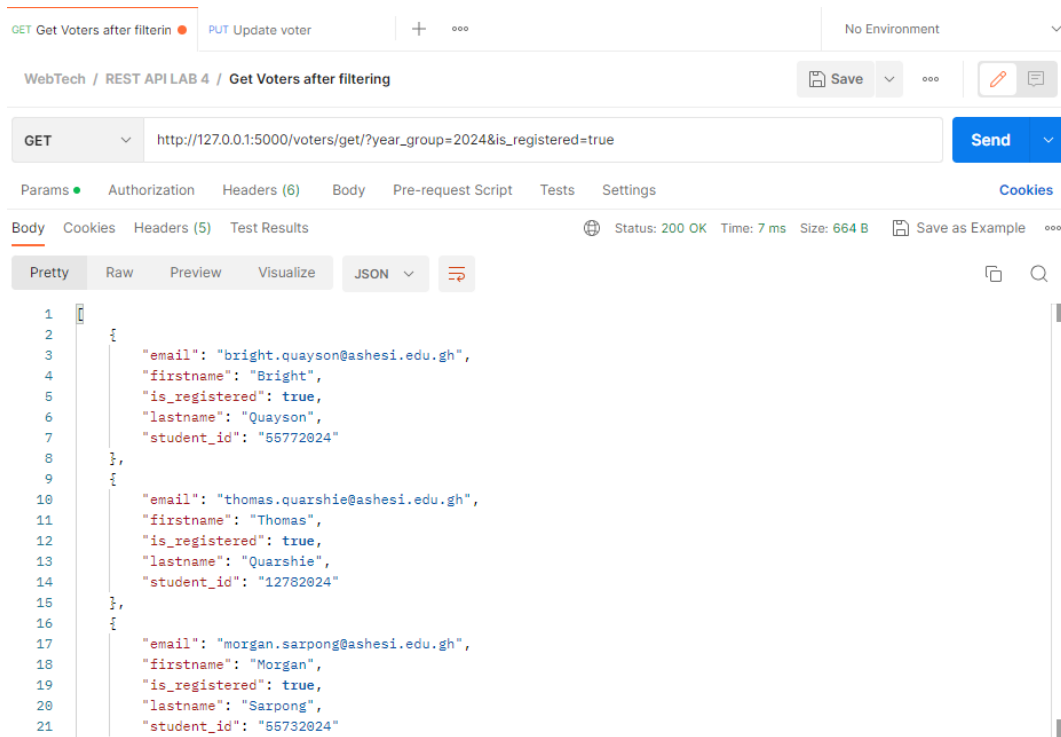


Figure 27 shows a GET request for retrieving voter information when the year_group is specified and is_registered is true (only registered voters).

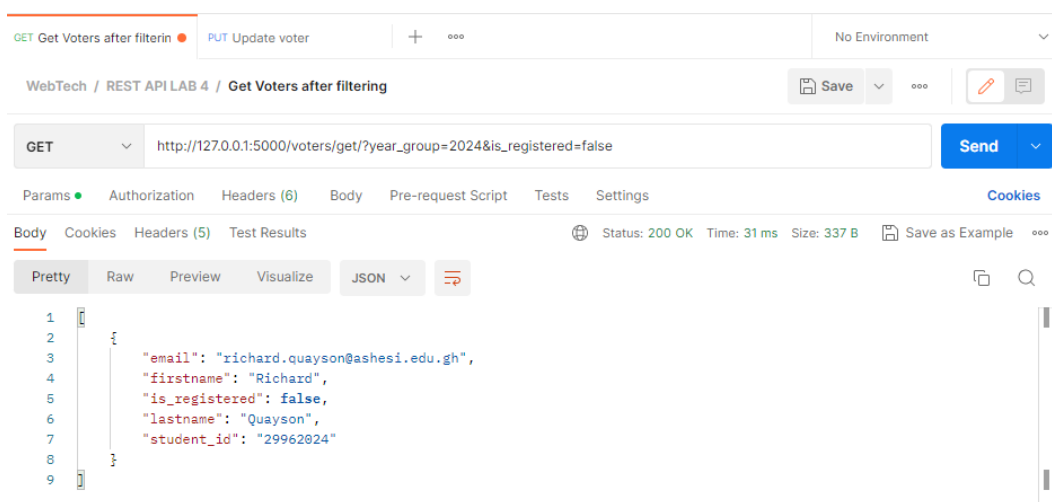


Figure 28 shows a GET request for retrieving voter information when the year_group is specified and is_registered is false (only de-registered voters).

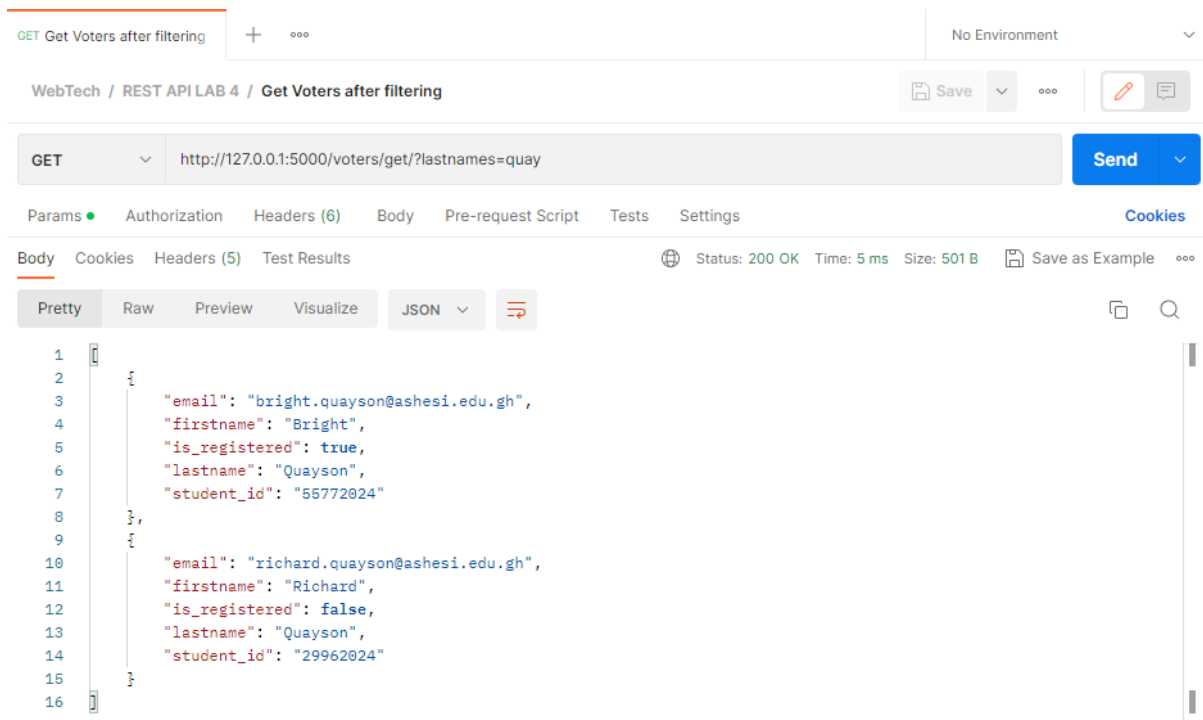


Figure 29 shows a GET request for retrieving voter information when the lastname of a voter starts with provided value.

5. Create an election.

Note: All validation of request such as empty body and required keys, from previous test with voters have been applied.

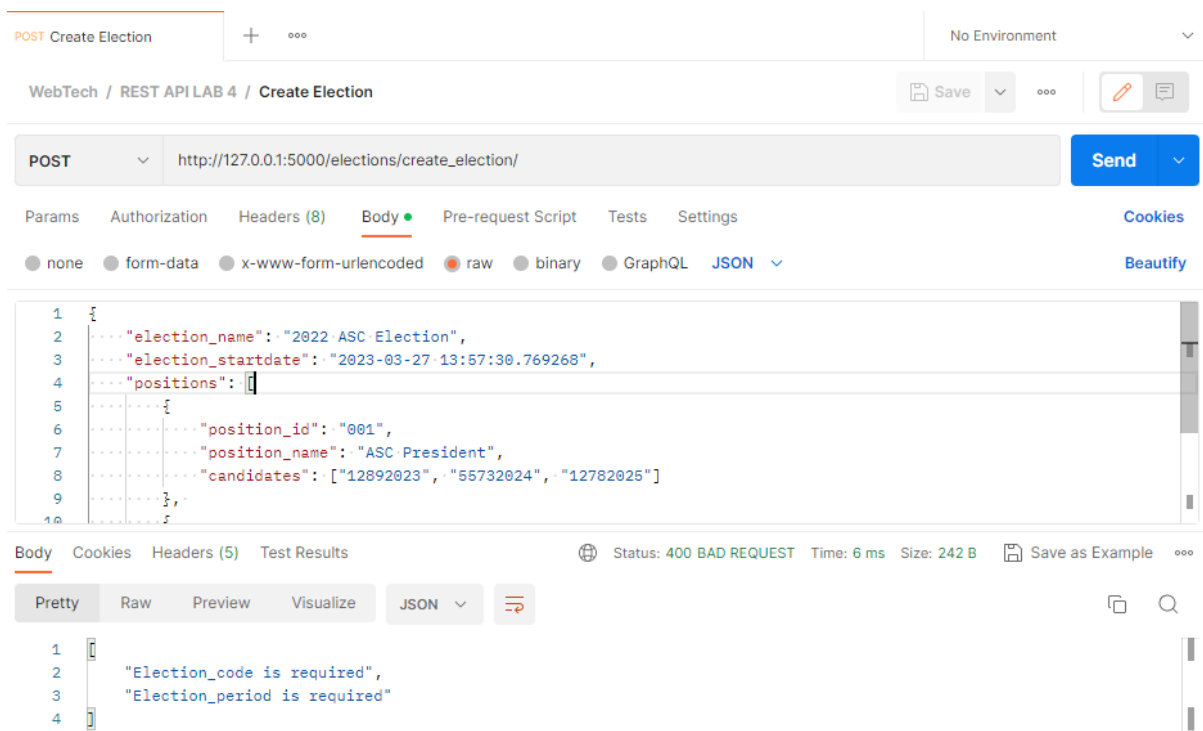


Figure 30 shows a POST request for creating an election when a required key is missing.

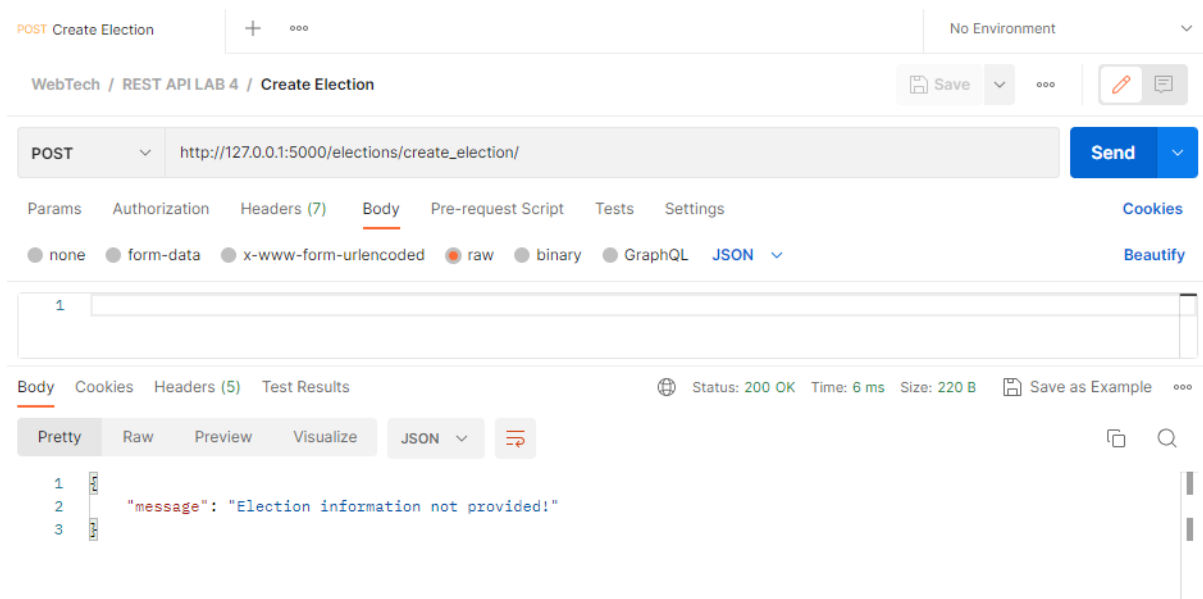


Figure 31 shows a POST request for creating an election when no information is provided in request body.

When an election unique constraint fails, the API returns a bad request with the appropriate message of the key whose constraint failed.

Note: current implementation only checks election_code and election name for uniqueness. Candidates' validity and other such validation have been assumed for this version of the API.

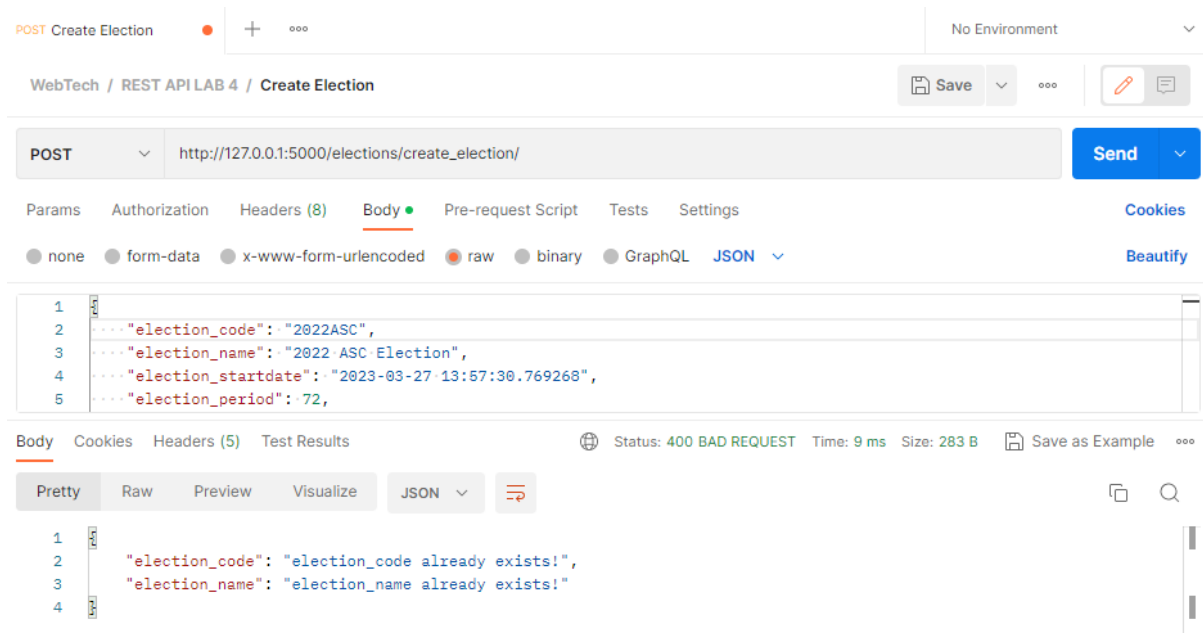


Figure 32 shows a POST request for creating an election when unique constraints fails.

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:5000/elections/create_election/`. The request body is a JSON object with the following structure:

```
1 {
2   "election_code": "2023ASC",
3   "election_name": "2023 ASC Election",
4   "election_startdate": "2023-03-27 13:57:30.769268",
5   "election_period": 72,
6   "positions": [
7     {
8       "position_id": "001",
9       "position_name": "ASC President",
10      "candidates": ["12892023", "55732024", "12782025"]
11    },
12    {
13      "position_id": "002",
```

The response status is 200 OK, with a time of 7 ms and a size of 1.09 KB. The response body is shown in JSON format:

```
1 {
2   "election_code": "2023ASC",
3   "election_name": "2023 ASC Election",
4   "election_period": 72,
5   "election_startdate": "2023-03-27 13:57:30.769268",
```

Figure 33 shows a POST request for creating an election when all constraints are passed.

6. Retrieve an election.

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:5000/elections/get/2023ASC/`. The response status is 200 OK, with a time of 7 ms and a size of 1.09 KB. The response body is shown in JSON format:

```
1 {
2   "election_code": "2023ASC",
3   "election_name": "2023 ASC Election",
4   "election_period": 72,
5   "election_startdate": "2023-03-27 13:57:30.769268",
6   "positions": [
7     {
8       "candidates": [
9         {
10          "candidate_id": "12892023",
11          "candidate_voters": []
12        },
13        {
14          "candidate_id": "55732024",
15          "candidate_voters": []
16        },
17        {
18          "candidate_id": "12782025",
19          "candidate_voters": []
20        }
21      ]
22    },
23    {
```

Figure 34 shows a GET request for retrieving election information when a valid election code is specified.

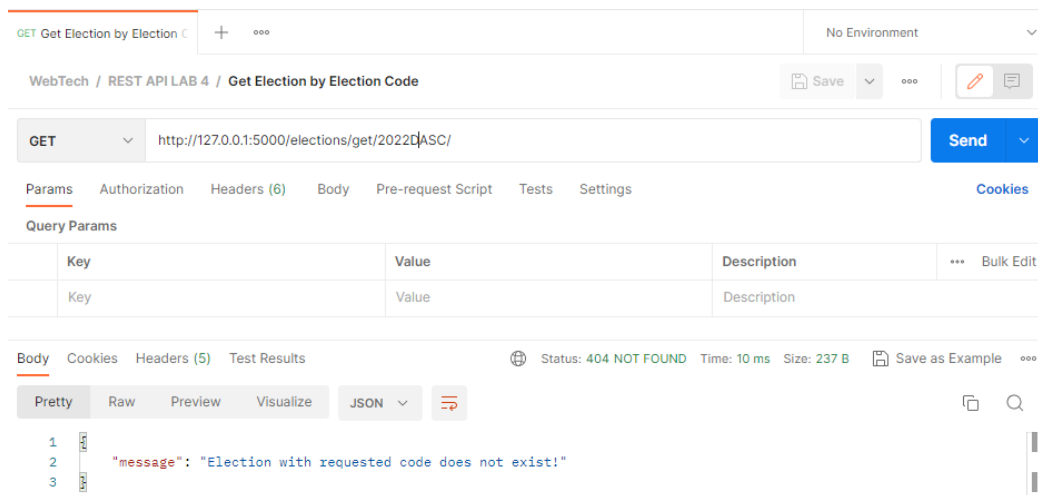


Figure 35 shows a GET request for retrieving election information when an invalid election code is specified.

7. Delete an election.

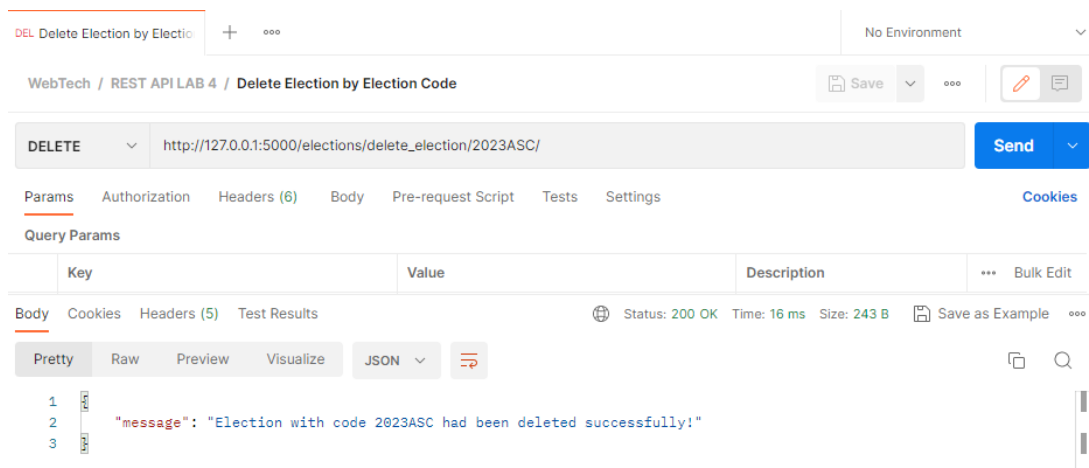


Figure 36 shows a DELETE request for deleting an election information when a valid election code is specified.

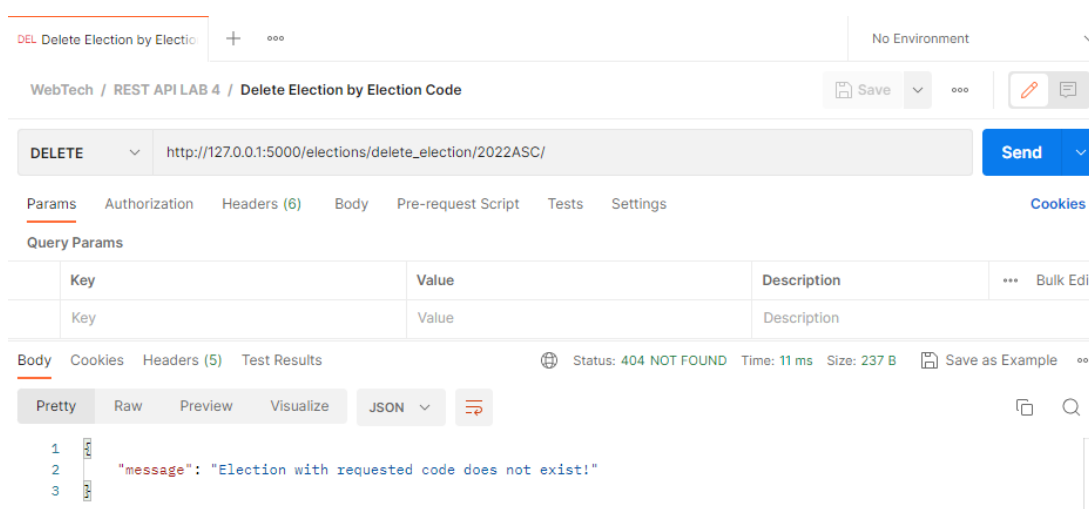


Figure 37 shows a DELETE request for deleting an election information when an invalid election code is specified.

8. Vote in an election

When the position the voter is casting their vote for is not specified, the API returns a bad request with the appropriate message that the election position is missing.

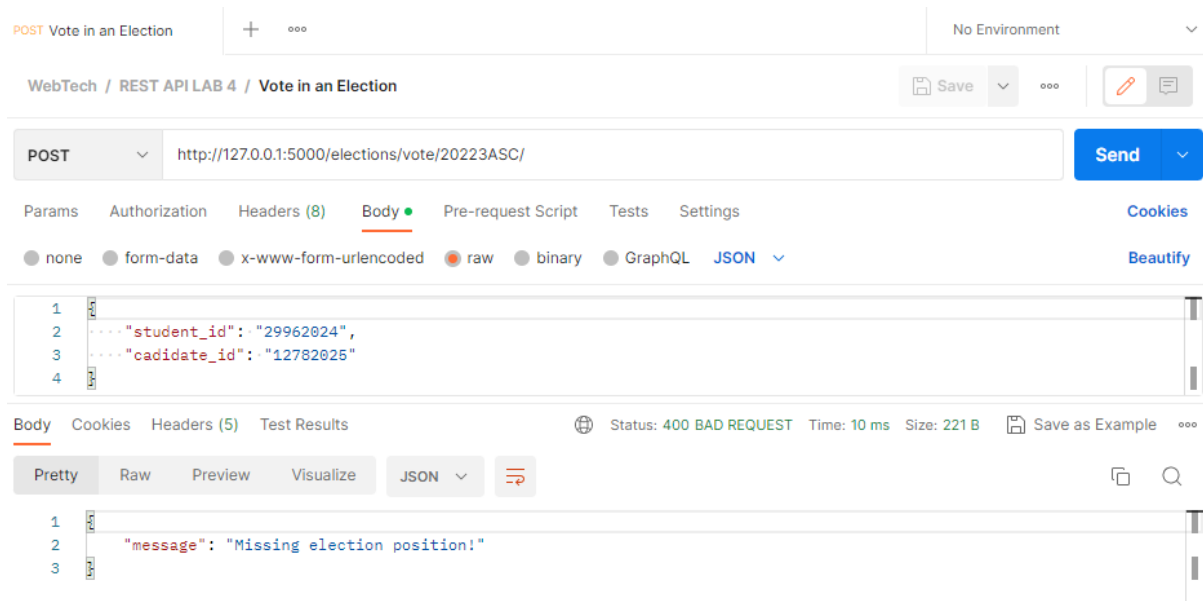


Figure 38 shows a POST request for voting in an election when the election position (say President) is not specified in the URL.

When the candidate_id the voter is casting their vote for is not specified, the API returns a bad request with the appropriate message that the election position is missing.

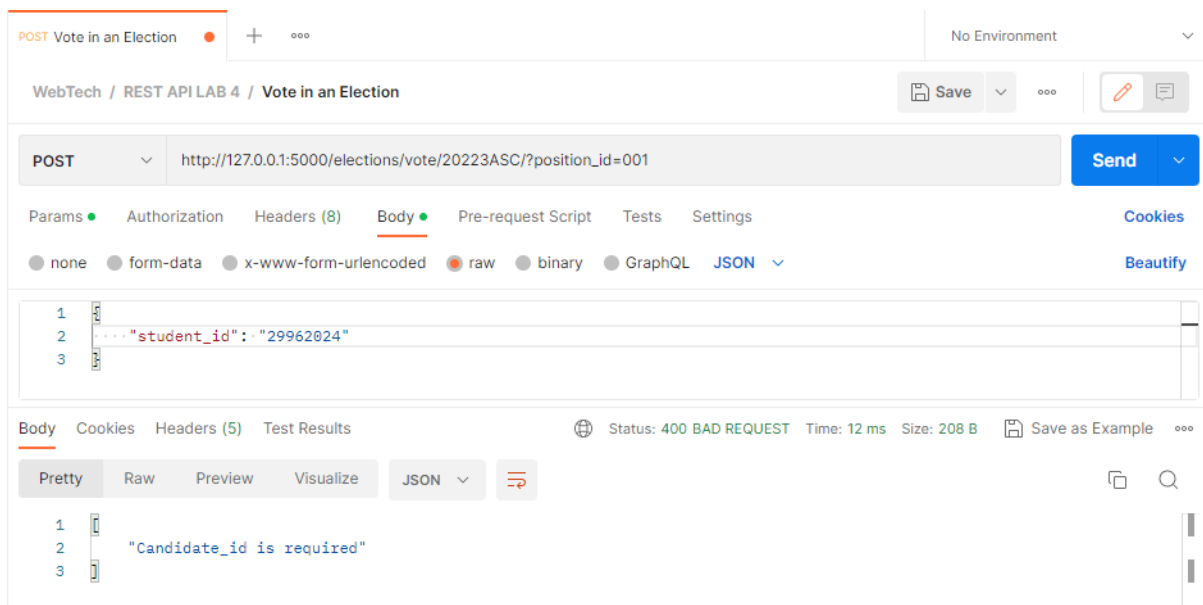


Figure 39 shows a POST request for voting in an election when the any expected key (in this case candidate_id) is not specified.

Note: All id validation has been applied here.

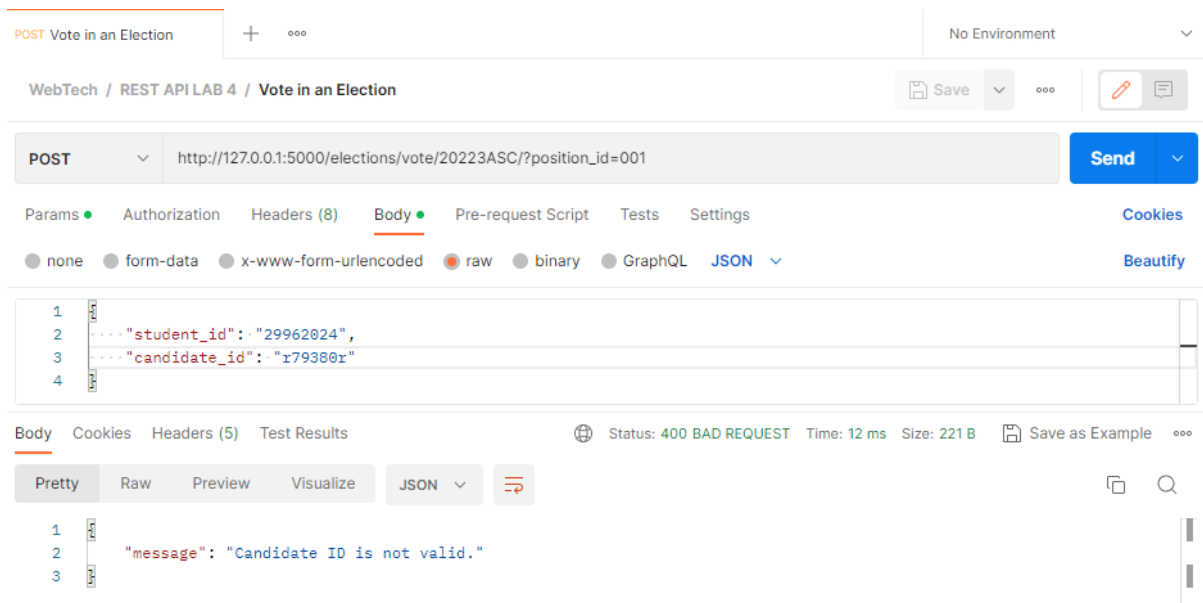


Figure 40 shows a POST request for voting in an election when an invalid `candidate_id` is specified.

When the voter or candidate specified hasn't been registered, the API returns a not found with the appropriate message that the voter or candidate has not been registered.

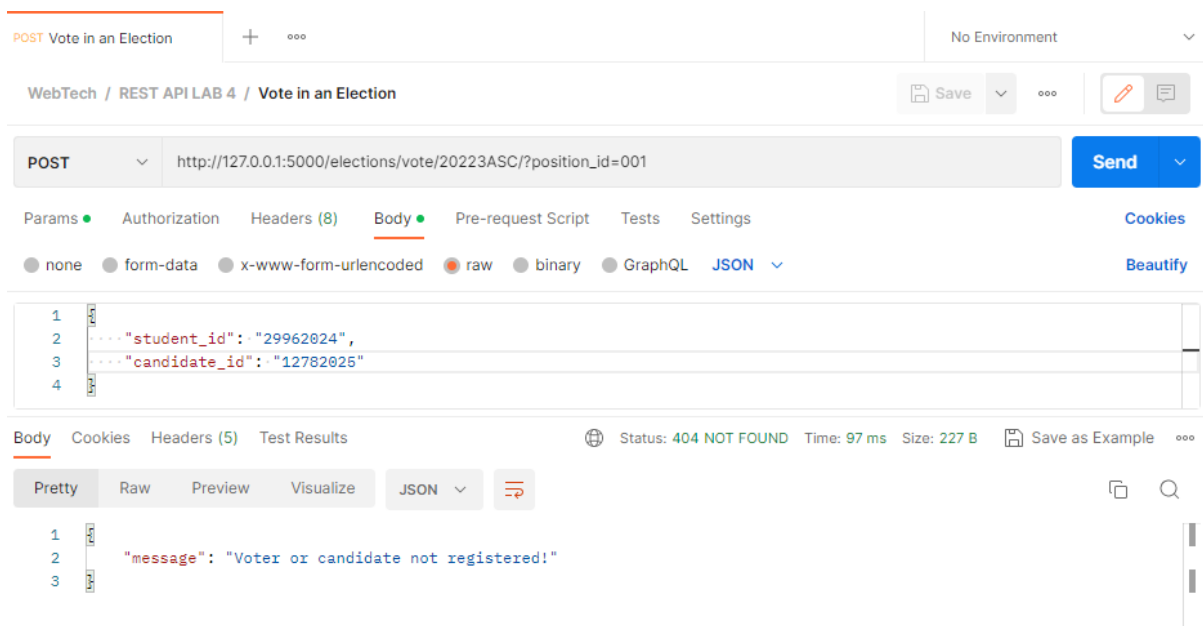


Figure 41 shows a POST request for voting in an election when the voter or candidate has not been registered.

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:5000/elections/vote/2022ASC/?position_id=001`. The request body is a JSON object: `{ "student_id": "55772024", "candidate_id": "12892023" }`. The response status is `404 NOT FOUND` with a message: `"message": "Election with code 2022ASC does not exist!"`. The interface includes tabs for Params, Authorization, Headers (8), Body, Pre-request Script, Tests, and Settings. The Body tab is active, showing the JSON body and the response message.

Figure 42 shows a POST request for voting in an election when the election has not been created.

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:5000/elections/vote/2023ASC/?position_id=001`. The request body is a JSON object: `{ "student_id": "55772024", "candidate_id": "12892023" }`. The response status is `200 OK` with a large JSON body containing election details: `{ "election_startdate": "2023-03-27 13:57:30.769268", "positions": [{ "candidates": [{ "candidate_id": "12892023", "candidate_voters": ["55772024"] }, { "candidate_id": "55732024", "candidate_voters": [] }] }] }`. The interface includes tabs for Params, Authorization, Headers (5), Body, Pre-request Script, Tests, and Settings. The Body tab is active, showing the JSON body and the response message.

Figure 43 shows a POST request for voting in an election when all constraints are passed.