**Today’s Agenda**

1. **Improving The Vote Class**
2. **Defining VotingService Class**
3. **Defining VoteRequestDTO Class**
4. **Defining The VoteController Class**
5. **Testing The API**
6. **Improving The Vote Class**
7. **Here also** we will **not return** the complete **Voter** and **Candidate** objects .
8. So to **drop them** from the **generated response** we would use **@JsonIgnore** with them
9. **But we must return *their id’s*** along with the **Vote** entity so that it becomes clear **who casted the vote** and to **whom**.
10. **This can be done** using **@JsonProperty** annotation.

**What is @JsonProperty annotation?**

The **@JsonProperty** annotation in **Jackson** is used to **control JSON property names** during **serialization** (Java object → JSON) and **deserialization** (JSON → Java object).

**Code For Vote Class**



**Now** the **response** would be like:

**{**

**"id": 1,**

**"voterId": 3,**

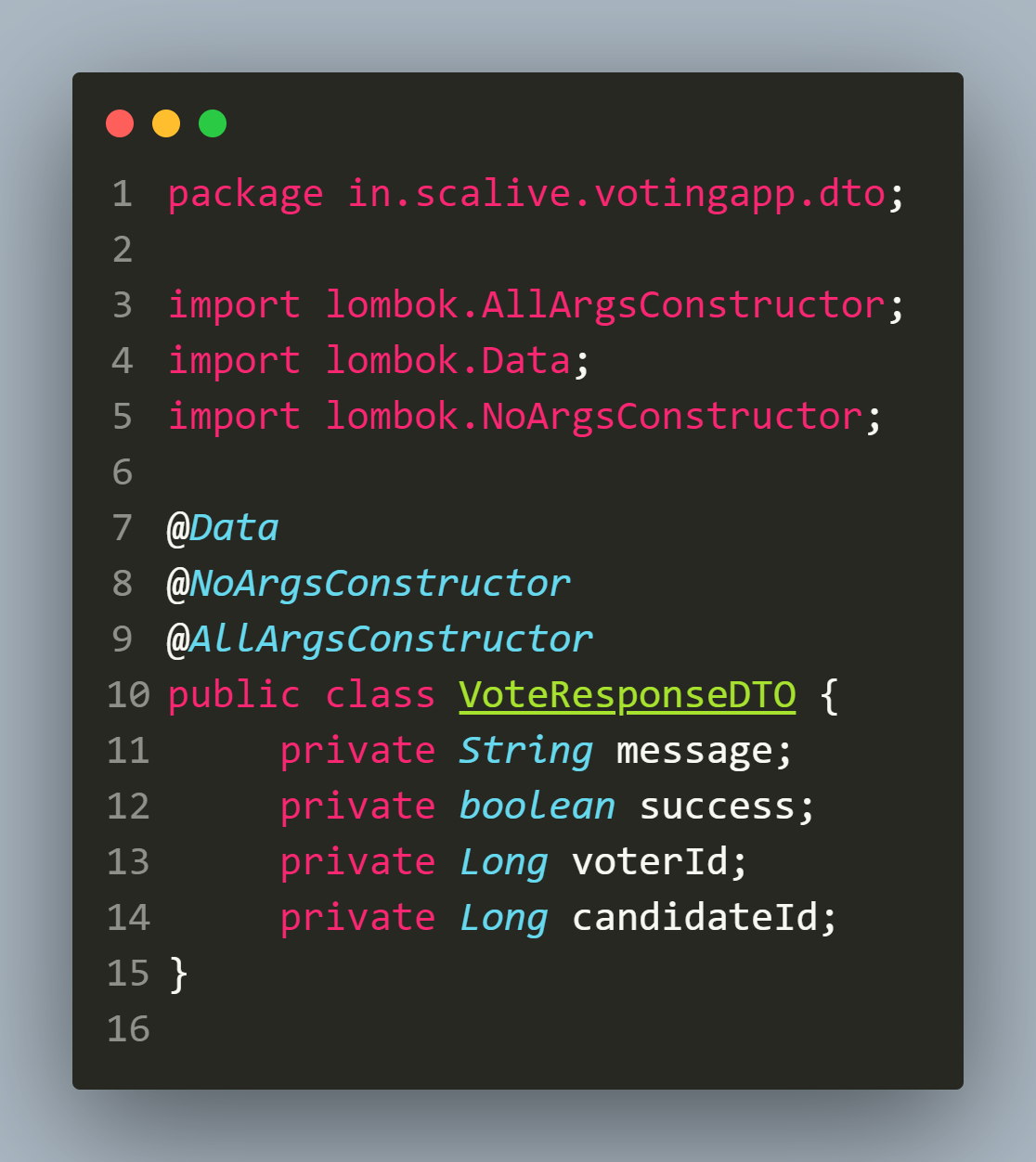
**"candidateId": 5**

**}**

1. **Defining The VoteResponseDTO Class**

1. This **class** is **needed** because when the **castVote()** method will run it will return **message**,**success** ,**voterId** and **candidateId** as response

2. **All of this** will be **returned** as a **VoteResponseDTO** object.



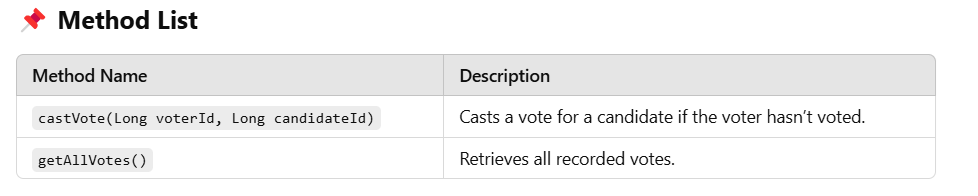
1. **Defining The VotingService Class**

📌 **Explanation**

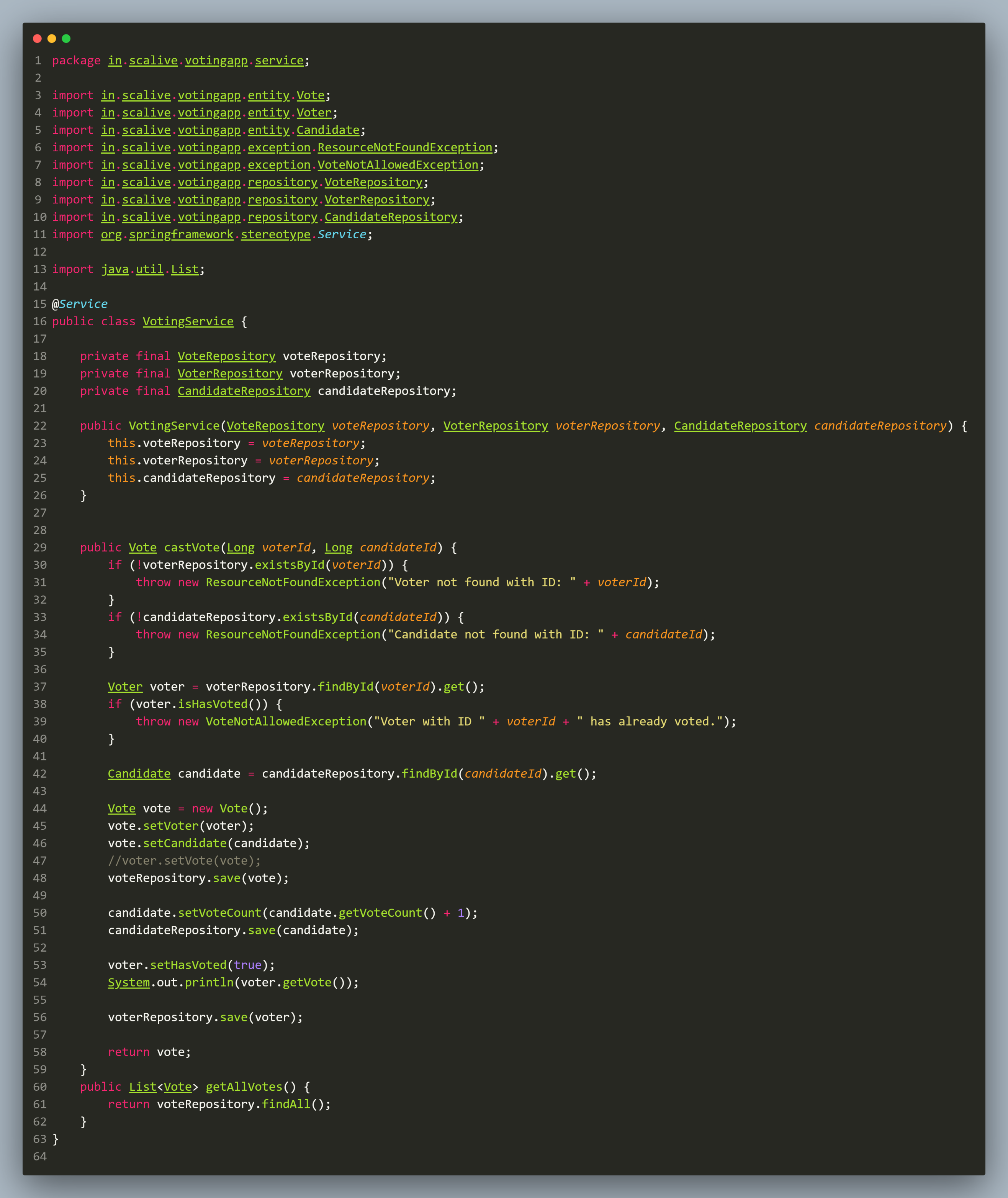
**Purpose:** Implements **business logic** for **handling voting operations**, including:  
 ✔ **Casting a vote** (**Ensures voters can vote only once**)  
 ✔ **Retrieving all votes**

📌 **Annotations Used**

* **@Service** → **Marks** this **class** as a **Spring service component**.



**Code For VotingService Class**



**Special Note**

**Qn: Why we are calling voteRepository.save(vote) explicitly ? I mean when we are already calling the method voterRepository.save(voter) won’t it save vote object also ?**

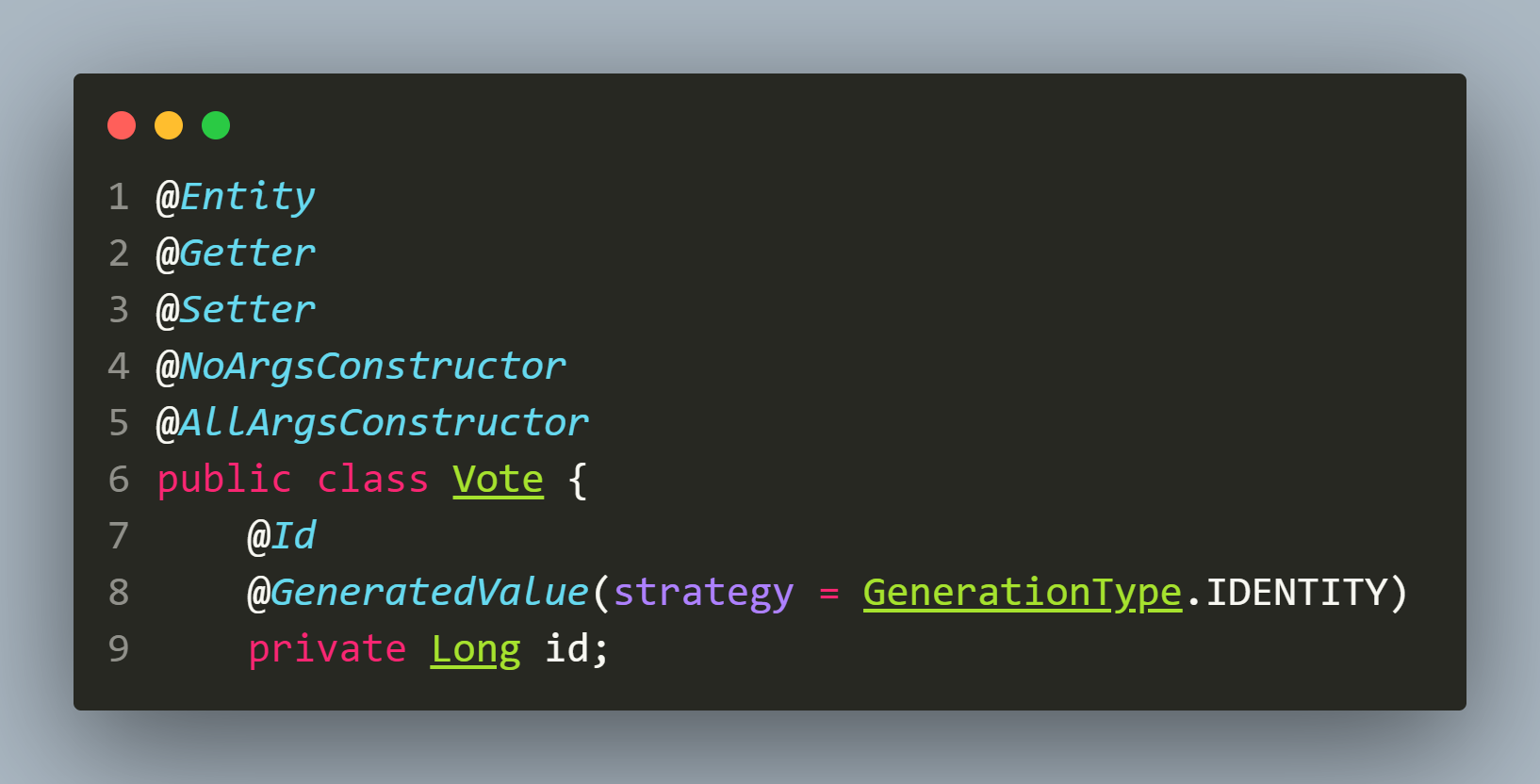
Ans: **No that’s because** the **voter** object **has no idea** about the **vote** object since **we have not called** the method **voter.setVote(vote)** . So it is **compulsory** for us to call **voteRepository.save(vote)** explicitly so that **vote** object also gets **saved**.

**Qn: How to avoid this extra repository call ?**

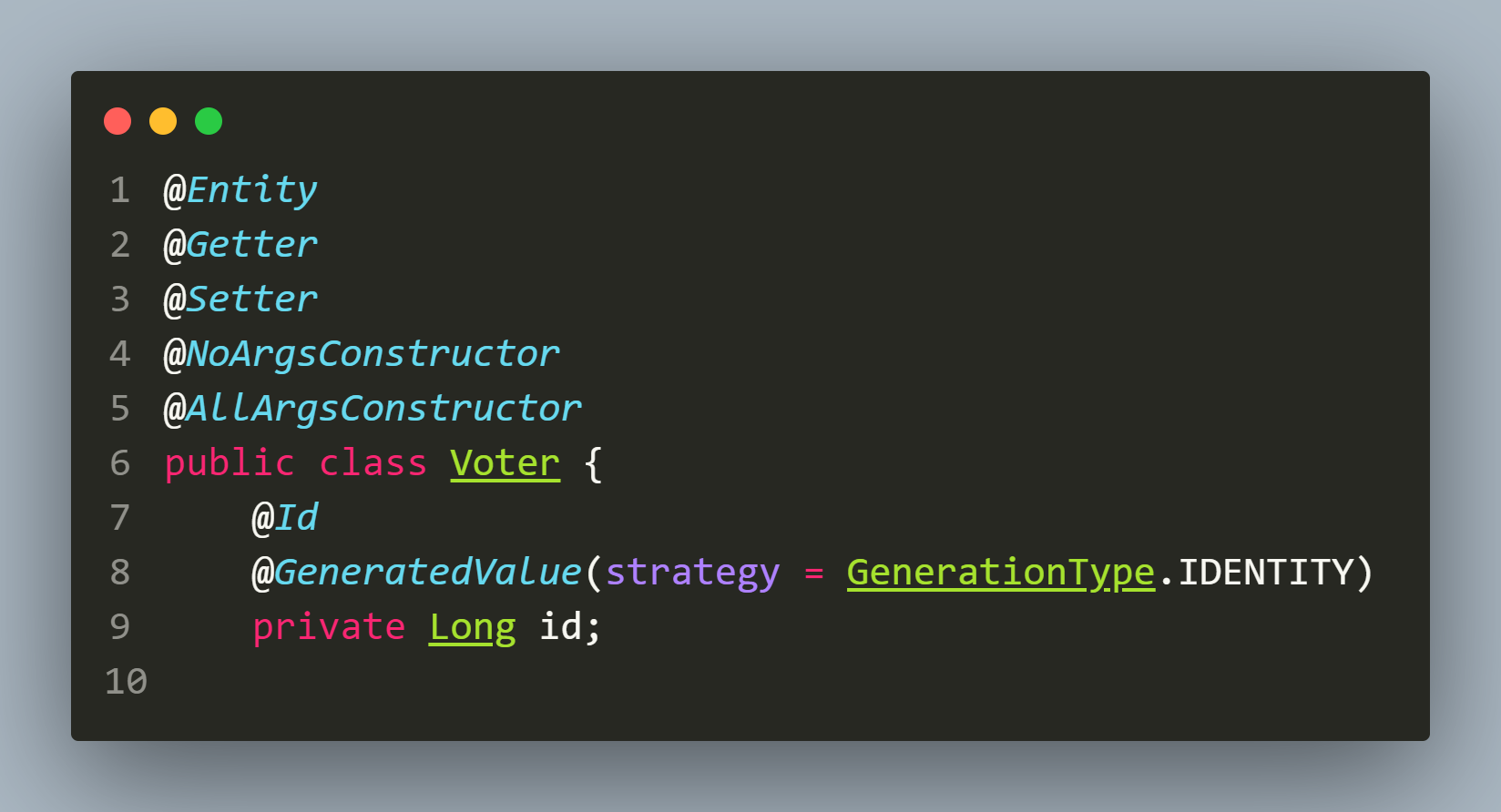
Ans: **In case** we want to **avoid** this then we need to **make following changes** in the code:

* + - 1. Set **vote** object in **voter** object by calling **voter.setVote(vote)** so that **voter** object becomes **aware** about **new vote object.**
      2. Replace **@Data** and use **@Setter**, **@Getter**, **@NoArgsConstructor** and **@AllArgsContructor** in **Vote** and **Voter** class. This is because **@Data** generates **toString(), equals(), hashCode(),** and all **getters/setters** — and in **bidirectional relationships**, it can **cause issues**.
      3. **Finally** just call **voterRepository.save(voter)** and it will save **Voter** and **Vote** both the objects.

**Modified Vote Class**



**Modified Voter Class**

****

**Modified castVote() Method**



1. **Defining The VoteRequestDTO Class**

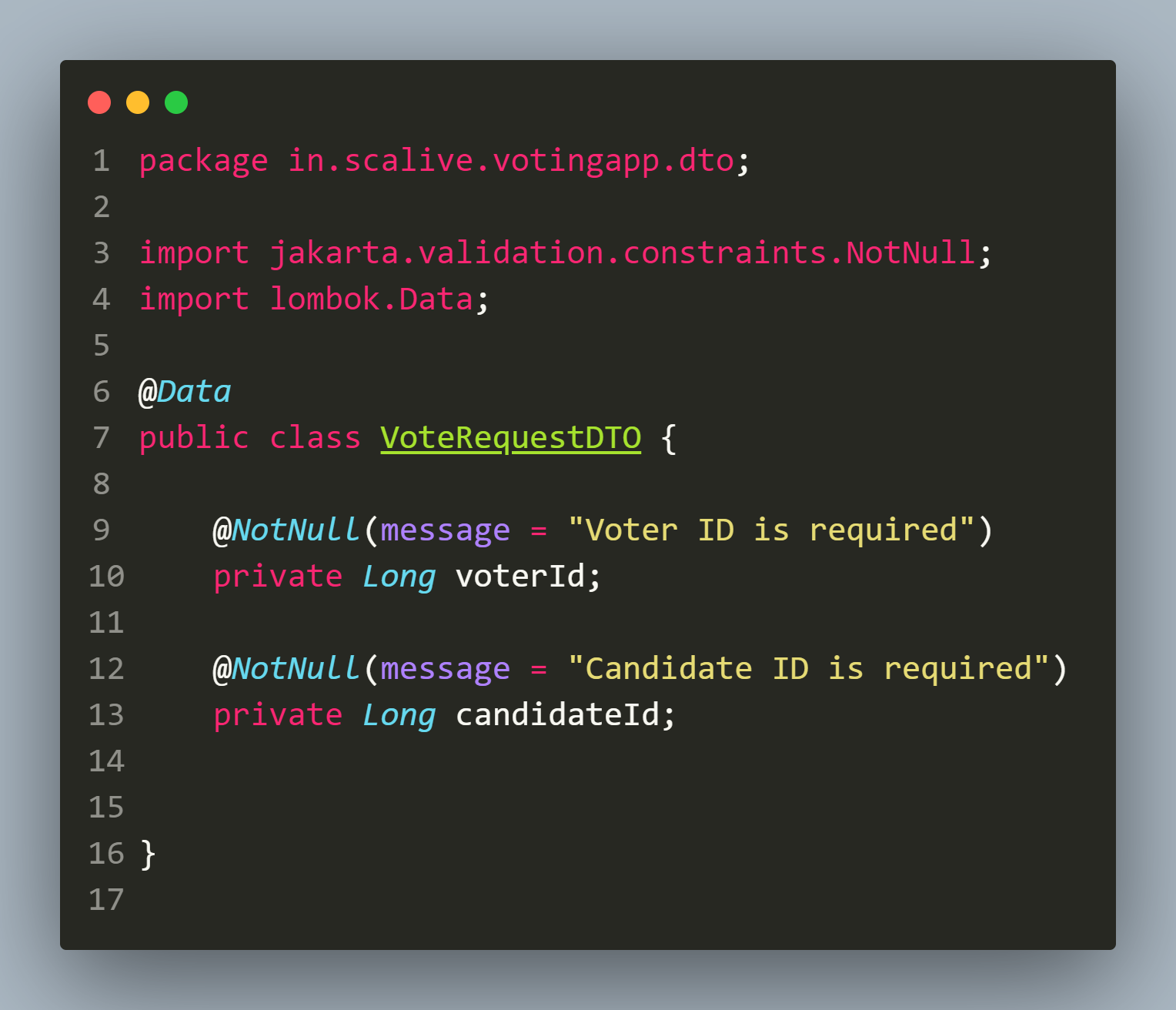
1. This **class** is **needed** because when the **vote** will be **casted** then **two values** will be **passed** which are **voterId** and **candidateId**

2. So **either we will have to pass them** as **query parameters** or as **object** in **request body**

3. Now since **casting a vote** will be a **POST request** so it is **recommended practice** to **pass this data** as **object** in **request body**.

4. This **object** will be of **VoteRequestDTO** class and will have just **2 fields** : **voterId** and **candidateId**.

**Code For VoteRequestDTO Class**



1. **Defining The VotingController Class**

📌 **Explanation**

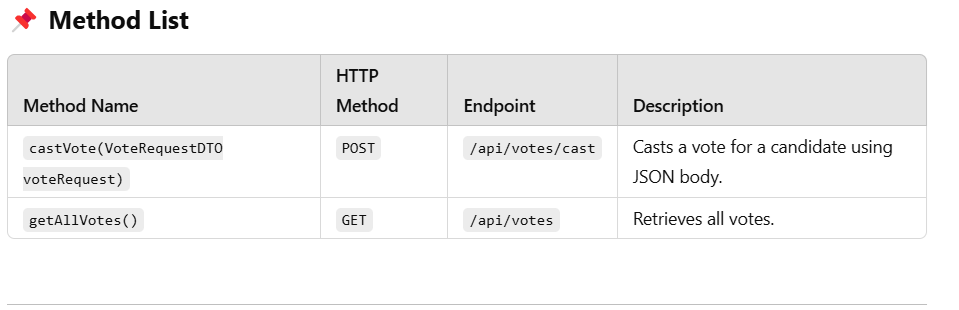
**Purpose:**

**Exposes REST API endpoints** for managing **voting operations**.

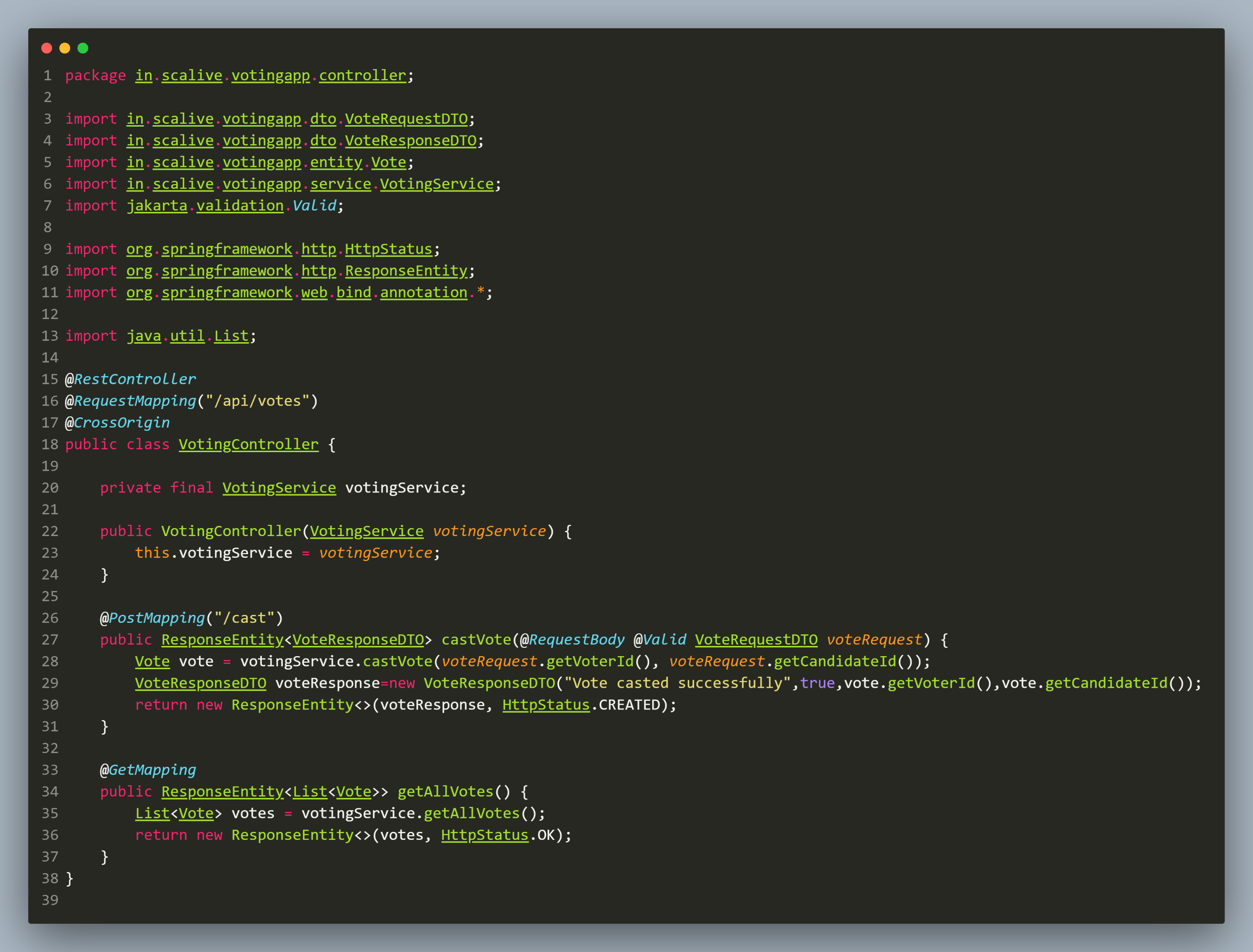
**Accepts JSON request body** instead of **query parameters** for **casting votes**.

📌 **Annotations Used**

* **@RestController** → **Marks** this **class** as a **REST API controller**.
* **@RequestMapping("/api/votes")** → **Base path** for all **vote-related APIs**.
* **@PostMapping, @GetMapping** → **Defines HTTP endpoints** for **voting operations**.
* **@RequestBody** → Accepts **JSON input** in the form of **VoteRequestDTO** object

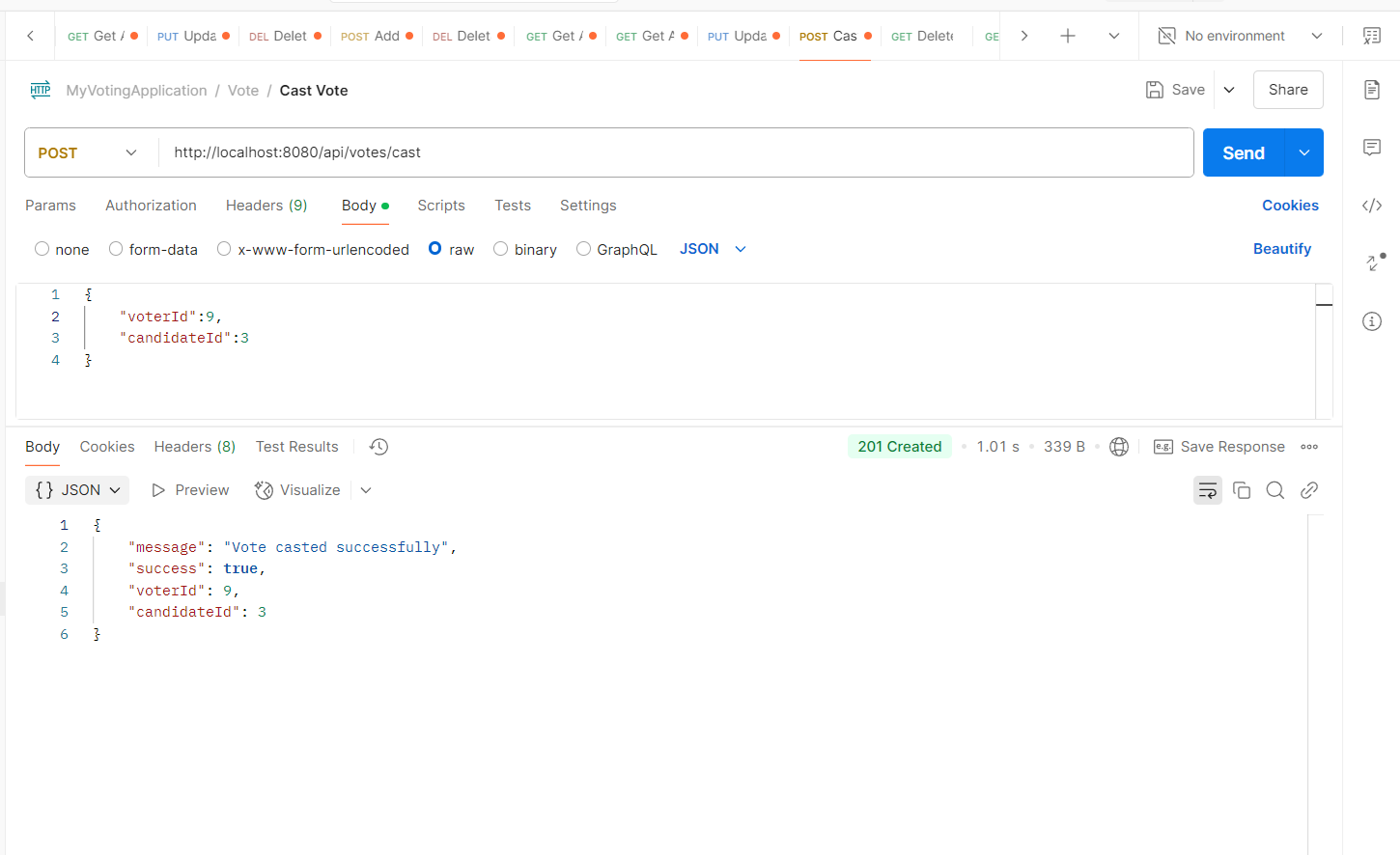


**Code For VotingController Class**

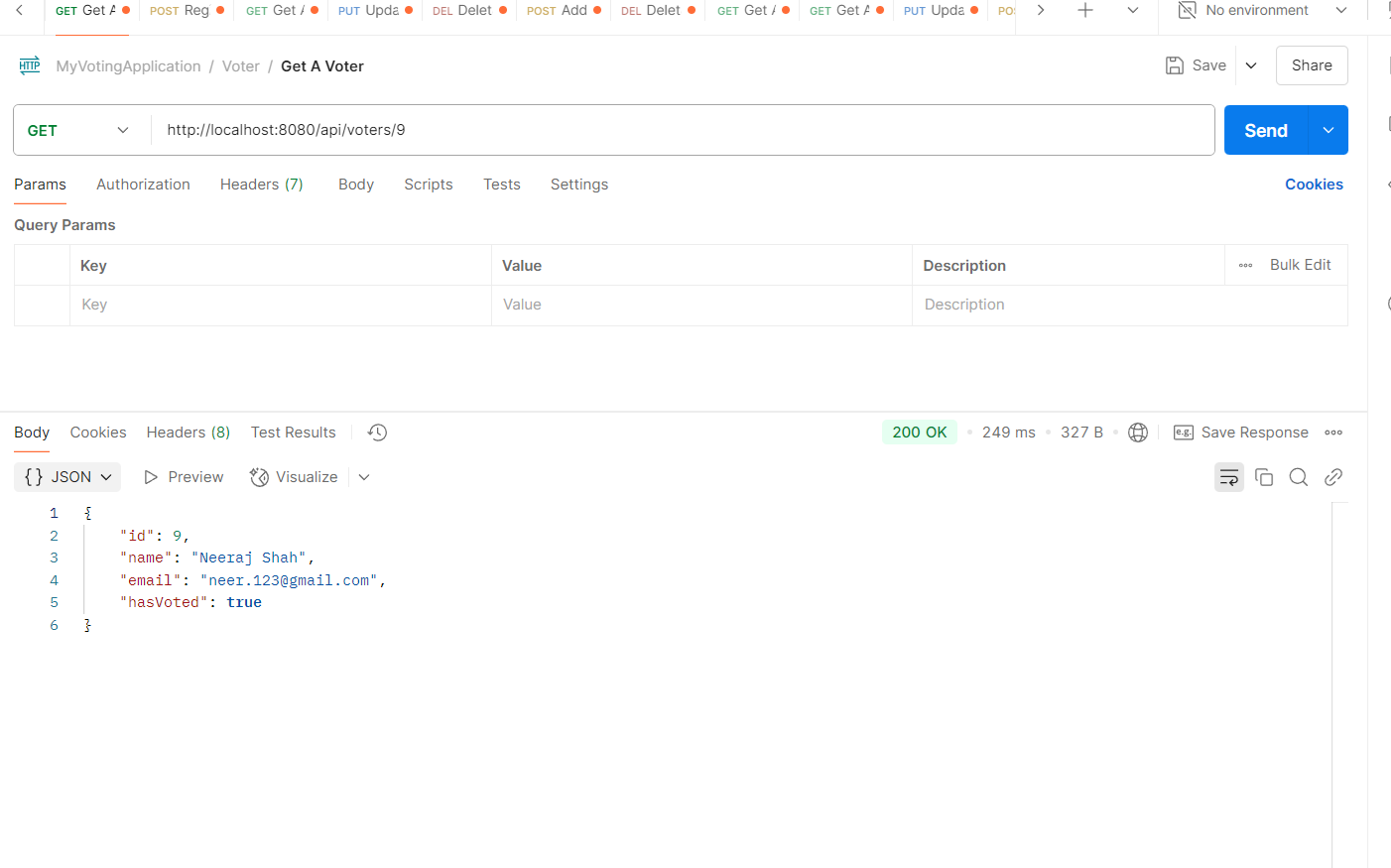


**5.Testing The Vote API**

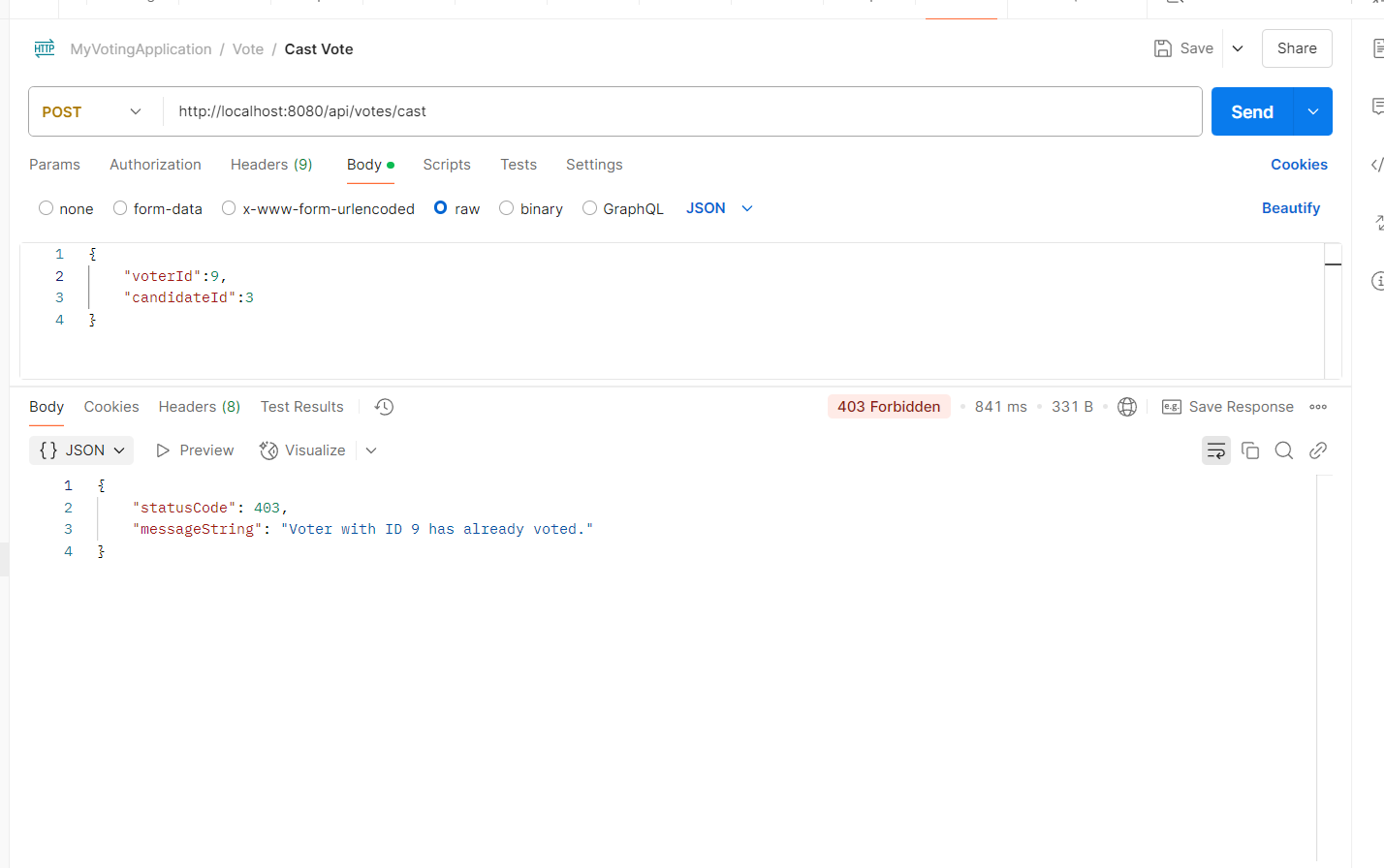
1. **Add Candidate: *http://localhost:8080/api/votes/cast***

****

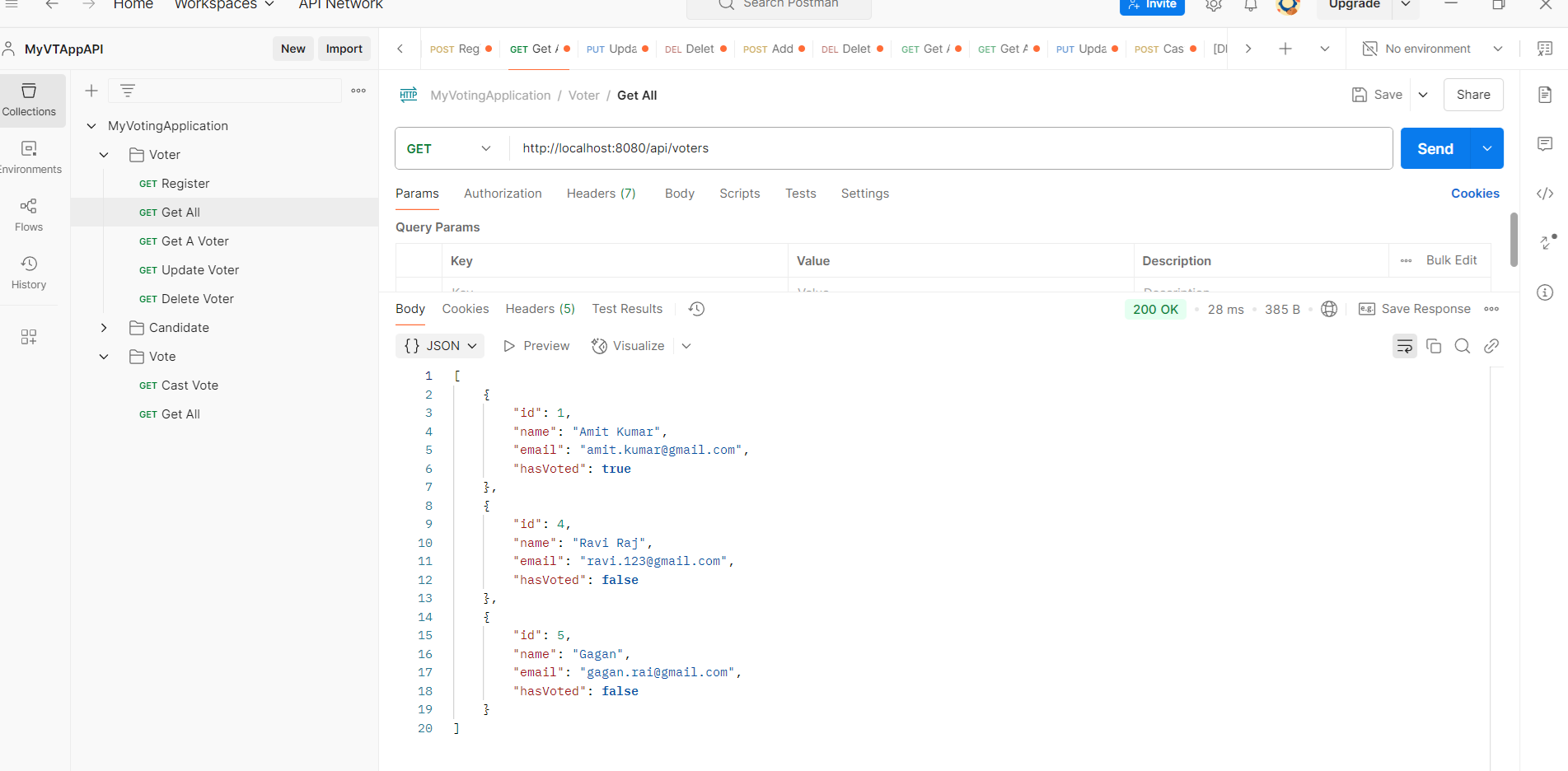
**2.Cross Checking Through Voter:** [***http://localhost:8080/api/voter/9***](http://localhost:8080/api/voter/9)

******

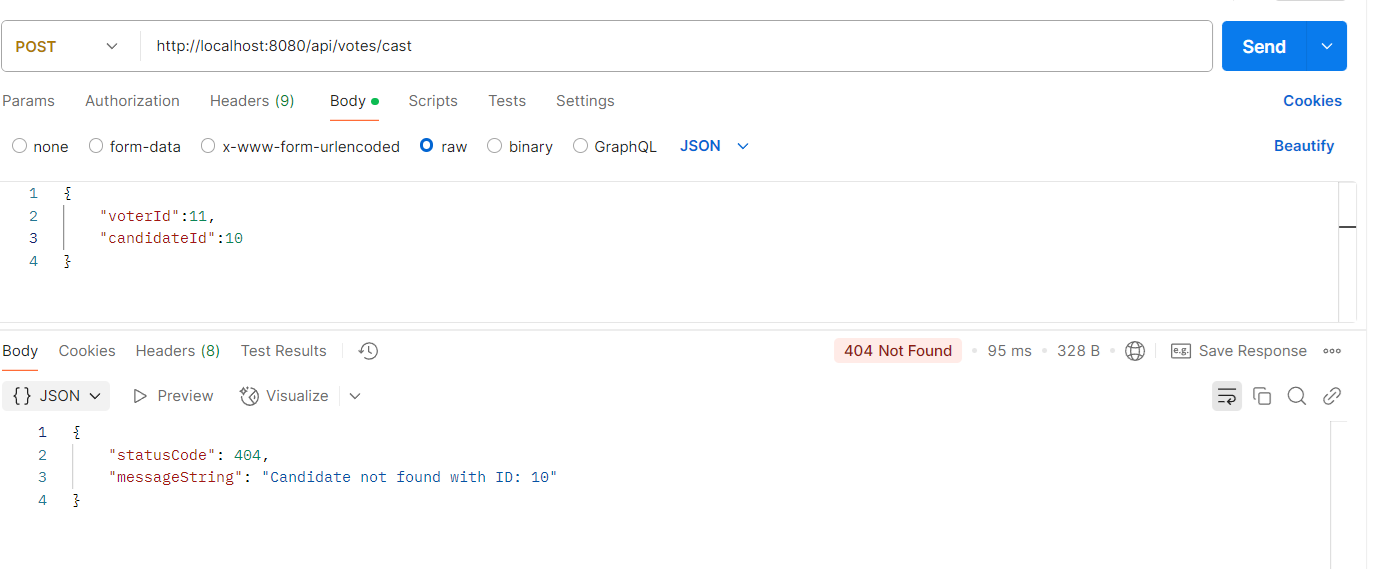
**3.Revoting By Same Voter:** [***http://localhost:8080/api/votes/cast***](http://localhost:8080/api/votes/cast)



**4.Cross Checking Voting Status Of All Voters:** [***http://localhost:8080/api/voters***](http://localhost:8080/api/voters)

****

**5.Voting For A Non Existing Candidate:** [***http://localhost:8080/api/votes/cast***](http://localhost:8080/api/candidates/add)



1. **Getting Details Of All Votes Casted: *http://localhost:8080/api/votes***

