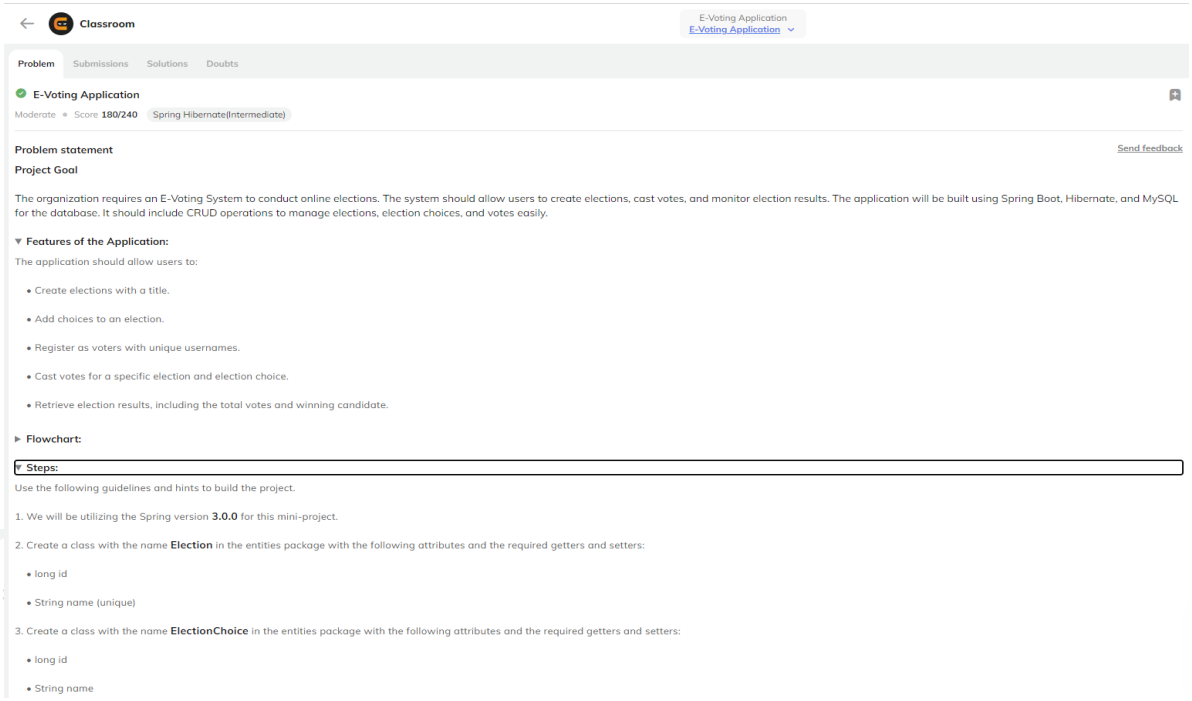
E-Voting Application

Problem Statement:



Classroom

E-Voting Application
E-Voting Application

Problem | Submissions | Solutions | Doubts

E-Voting Application

Moderate • Score 180/240 Spring Hibernate(Intermediate)

Problem statement

Send feedback

Project Goal

The organization requires an E-Voting System to conduct online elections. The system should allow users to create elections, cast votes, and monitor election results. The application will be built using Spring Boot, Hibernate, and MySQL for the database. It should include CRUD operations to manage elections, election choices, and votes easily.

▼ Features of the Application:

The application should allow users to:

- Create elections with a title.
- Add choices to an election.
- Register as voters with unique usernames.
- Cast votes for a specific election and election choice.
- Retrieve election results, including the total votes and winning candidate.

▶ Flowchart:

▼ Steps:

Use the following guidelines and hints to build the project.

1. We will be utilizing the Spring version **3.0.0** for this mini-project.
2. Create a class with the name **Election** in the entities package with the following attributes and the required getters and setters:
   - long id
   - String name (unique)
3. Create a class with the name **ElectionChoice** in the entities package with the following attributes and the required getters and setters:
   - long id
   - String name
   - Election election (Many To One relationship with Election entity)
4. Create a class with the name **User** in the entities package with the following attributes and the required getters and setters:
   - long id
   - String name (unique)
5. Create a class with the name **Vote** in the entities package with the following attributes and the required getters and setters:
   - long id
   - User user (One To One relationship with User entity)
   - Election election (Many To One relationship with Election entity)
   - ElectionChoice electionChoice (Many To One relationship with ElectionChoice entity)
6. Define relationships between entities using annotations.
   - ElectionChoice - Election Class: Many-to-One Relationship
   - Vote - User Class: OneToOne relationship
   - Vote - Election Class: ManyToOne relationship
   - Vote - ElectionChoice Class: ManyToOne relationship
7. Create repository interfaces in the repositories package for each entity to handle database operations using Spring Data JPA.
8. Implement service classes in the service package for each entity to handle business logic.
9. Create REST API endpoints for creating elections, adding election choices, registering users, casting votes, and retrieving election results.
10. Implement the necessary CRUD operations for managing elections, candidates, voters, and votes.
11. Test the application using tools such as Postman to ensure data is saved and retrieved correctly from the database.

▼ End Points To Be Created:

1. Election Endpoints:

- **GET "/get/elections"**: It retrieves the list of all elections.
- **POST "/add/election" (@RequestBody Election election)**: It creates a new election.

2. ElectionChoice Endpoints:

- **POST "/add/electionChoice" (@RequestBody ElectionChoice electionChoice)**: It adds a election choice.
- **GET "/get/electionChoices"**: It fetches the list of all electionChoices from the database.
- **POST "/count/election/choices" (@RequestBody Election election)**: It retrieves the list of all choices for the given election.

3. User Endpoints:

- **POST "/add/user" (@RequestBody User user)**: It creates a new user into the database.
- **GET "/get/users"**: It fetches the list of all users from the database.

4. Vote Endpoints:

- **GET "/get/votes"**: It fetches the list of all votes from the database.
- **POST "/add/vote" (@RequestBody Vote vote)**: It register a new vote into the database.
- **GET "/count/votes"**: It fetches the count of total votes from the database.
- **POST "/count/election/votes" (@RequestBody Election election)**: It returns the count of total votes by-election in long.

5. Results Endpoints:

- **POST "/winner/election" (@RequestBody Election election)**: It retrieves the ElectionChoice for a specific election as the winner.

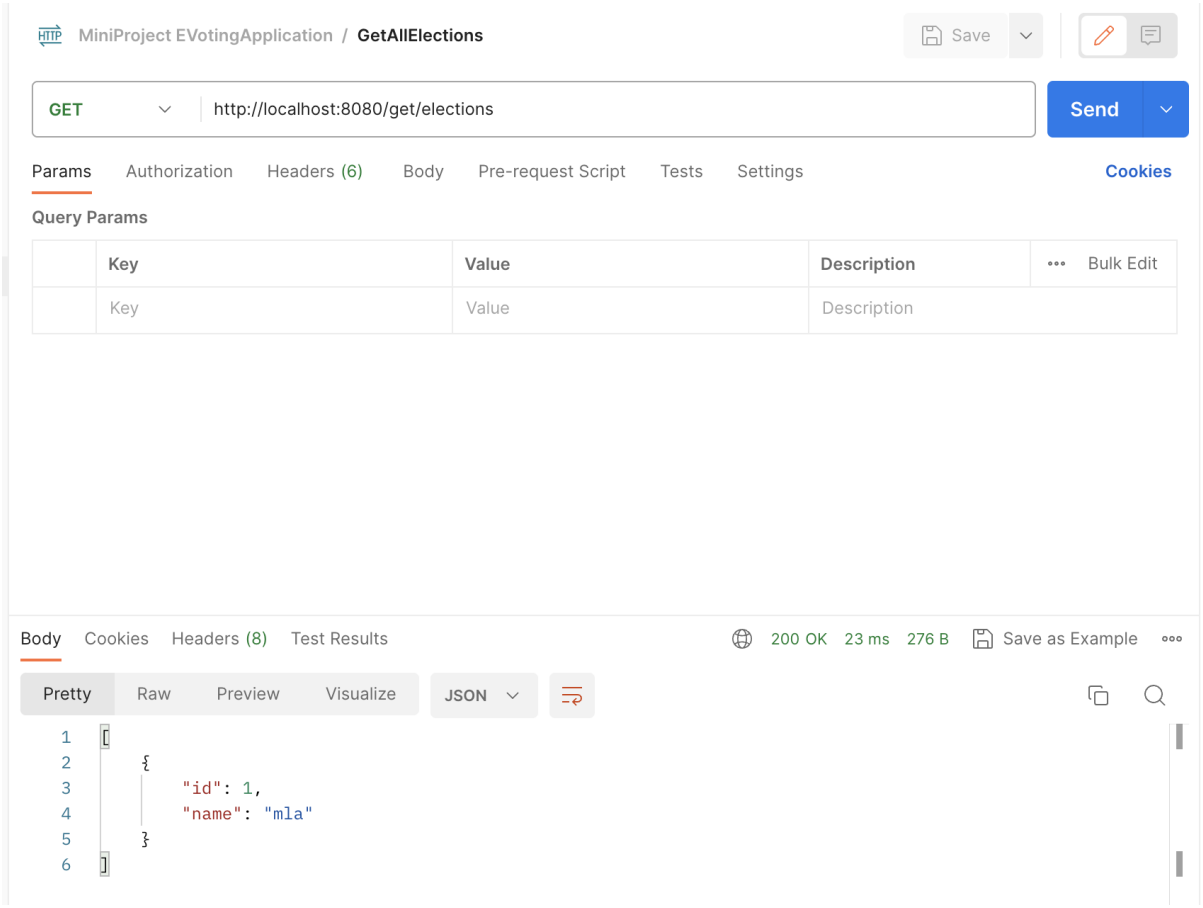
Postman Testing:

▼ Testing on Postman:

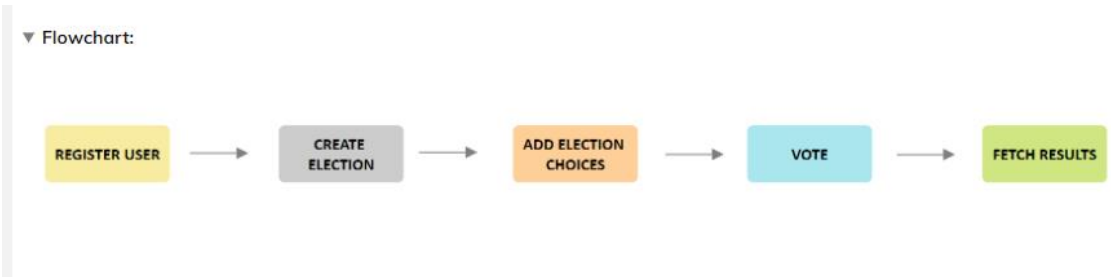
After successfully creating the application, you need to test its functionality. Your application should be tested for the following scenarios:

- Creating an election: The application should successfully store the election details in the database.
- Adding Election Choices: Election Choices should be associated with the corresponding election.
- Registering a user: The application should store the voter details.
- Casting a vote: The application should record the vote for the specified election and election choice.
- Retrieving election results: The application should retrieve the total votes and winning choice for a specific election.
- Test all the exposed APIs of the backend on Postman to ensure proper functionality and data handling.

▼ Reference image


MiniProject EVotingApplication / GetAllElections
Save
GET
http://localhost:8080/get/elections
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
200 OK
23 ms
276 B
Save as Example
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"name": "mla"
}
]

Flow Chart:


Flowchart:
REGISTER USER
CREATE ELECTION
ADD ELECTION CHOICES
VOTE
FETCH RESULTS