President Eletction Application Documentation

Martynas Galkinas

2023 m. lapkričio 13 d.

1 Project

1.1 Requirement Specification

Conditions:

- There are multiple (unlimited number) candidates. You can have those predefined.
- Voter can vote for only one of them.
- No voter can change his/her decision once submitted.
- There is no need to keep track of any voter data except and identifier, region and their vote
- Assume that customer is identified by third party service, do not implement registration or authentication.

Tasks:

1. Implement an endpoint that returns a list of candidates available: name, number on the list, short summary of their agenda.
2. Implement an endpoint that enables voting for the candidate.
3. Implement endpoints that return voting result reports.
    - Overall distribution of votes amongst candidates.
    - Voting result distribution amongst different regions.
    - The winner endpoint. It must return a single candidate if he/she was voted for by more that 50%. Otherwise it must return two most voted candidates.

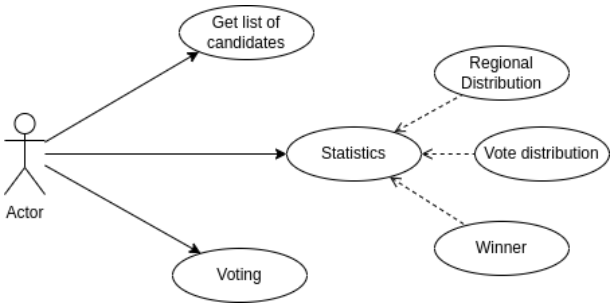
Requirements:

- You can use any of the following programming languages Java, Scala, Kotlin, Groovy
- You may use any frameworks you need.
- There are no specific requirements for data storage. You can keep it in memory.

- All interaction with an application must be implemented either as REST or GraphQL endpoints.
- Be mindful about naming and comments. Your core must be readable and clean.
- Your final delivery must be either Maven or Gradle Project.

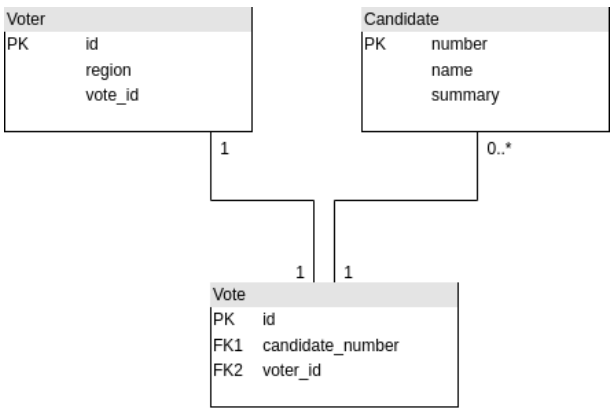
1.2 System project

1.2.1 Usage diagram



1 pav.: Use case diagram

1.2.2 Database diagram



2 pav.: Database diagram

1.2.3 API documentation

This is a subsection consisting of the API endpoint documentation.

API endpoint	GET /api/election/getCandidates
Expected response	[{"number": 1,"name": "Jonas","summary":"Uz geresne ateitį"}, {"number": 2,"name": "Petras","summary": "Už blogesnę praeitį"}, {"number": 3,"name": "Gryčius","summary": "Už Garstyčias grikiuose"}, {"number": 4,"name": "Margarita","summary": "Už mašinų draudimą keliuose"}, {"number": 5,"name": "Bertalomėja","summary": "Už <30kvm būtų naikinimą"}]
Possible response	[ ]

API endpoint	POST /api/election?voter={voter_id}&candidate={candidate_id}
Request Variables	{voter_id} - voter identification number {candidate_id} - candidates list number
Expected response	{"message": "Successfully submitted vote for candidate No. 1"}
Possible response	{"message": "Failed to submit vote"}

API endpoint	GET /api/election/statistics?type=total
Expected response	{"Margarita": 1,"Bertalomėja": 2,"Jonas": 7,"Gryčius": 3}
Possible response	[ ]

API endpoint	/api/election/statistics?type=regional

Expected response	{
	"Bertalomėja, Vilnius":1,
	"Jonas, Kaunas":3,
	"Jonas, Šiauliai":1,
	"Margarita, Kaunas":1,
	"Jonas, Klaipėda":1,
	"Bertalomėja, Klaipėda":1,
	"Gryčius, Vilnius":2,
	"Gryčius, Kaunas":1,
	"Jonas, Vilnius":2
	}

Possible response	[ ]

API endpoint	/api/election/statistics?type=winner
Expected response	{"Jonas": 7}
Possible response	{"Jonas": 5,"Gryčius": 4}, [ ]

2 Testing

2.1 Unit tests

Unit tests were written in JUnit 5 for the basic classes: Vote, Candidate, Voter.

2.2 Integration tests

Manual integration test were performed using Postman tool(the requests used for testing can be found in the misc folder of this repository).

3 Sources

Gitlab repository - https://gitlab.matrasas.dev/zeburgana/electionbackend