POLITECHNIKA ŚWIĘTOKRZYSKA

Wydział Elektroniki, Automatyki i Informatyki

SYSTEMY ODPORNE NA BŁĘDY **SEMESTR ZIMOWY, 2021 R.**

KIERUNEK:
INFORMATYKA

REALIZACJA:
ADRIAN JAKUBCZYK GR. 1ID21A
KAMIL SITEK GR. 1ID21A
JAROSŁAW SPYRKA GR. 1ID12A

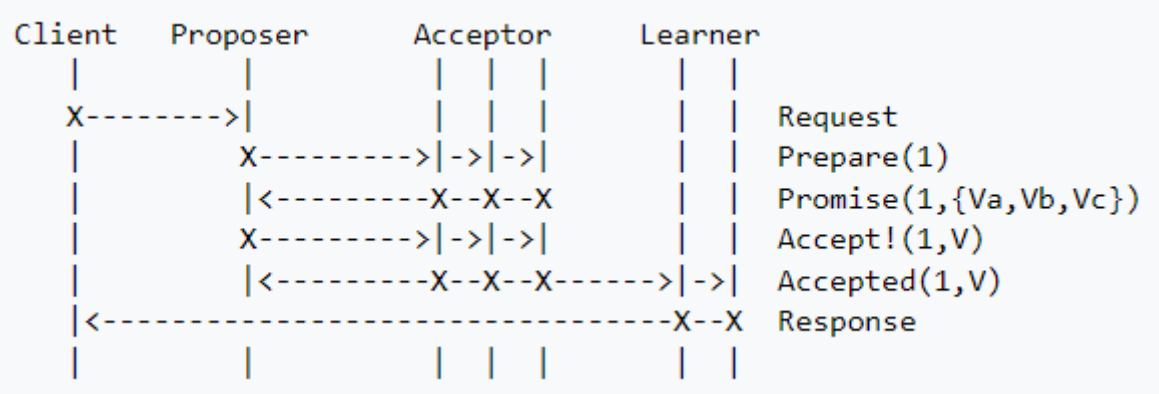
Paxos głosowanie

1. Wstęp teoretyczny.

Paxos głosowanie - zastosowanie algorytmu Paxos do głosowania nad wartością. System powinien składać się z 8 serwerów z zdefiniowanym liderem, algorytm ma na celu porozumiewanie się pomiędzy elementami systemu i ustalenie na drodze głosowania wspólnej wartości, a następnie jej zwrócenie.

Paxos jest rodziną protokołów pierwotnie przedstawioną przez Leslie Lamport w 1989 roku. Protokoły te rozwiązują problem konsensusu w asynchronicznej sieci zawodnych procesów. Paxos zapewnia stabilność nawet przy wielu błędach. Jest uważany za pierwszy protokół konsensusu, z matematycznym dowodem działania. Podstawowy protokół Paxos składa się z dwóch faz, które są dzielone na dwie podfazy:

- Prepare (Faza 1A): Proposer tworzy wiadomość identyfikowaną unikalnym numerem (n), który powinien być największym z użytych do tej pory.
- Promise (Faza 1B): Acceptor odbiera wiadomość od Proposera i sprawdza jej numer, jeśli numer jest większy od otrzymanego do tej pory, Acceptor zwraca Promise do Proposera
- Accept (Faza 2A): Proposer po otrzymaniu Promise od większości Acceptorów wysyła wiadomość z wartością którą chce przegłosować.
- Accepted (Faza 2B): Acceptor otrzymuje wiadomość Accept od Proposera i musi ją zaakceptować jeśli nie obiecywał już przyjęcia wniosku z wyższym identyfikatorem.



Paxos umożliwia wielu proposerom wysyłanie różnych sprzecznych wiadomości do akceptacji. Dzięki takiemu podziałowi na poszczególnych krokach może pojawić się błąd ale Paxos zapewnia że ostatecznie Acceptorzy zgadzają się na pojedynczą wartość.

2. Technologie.

Backend:

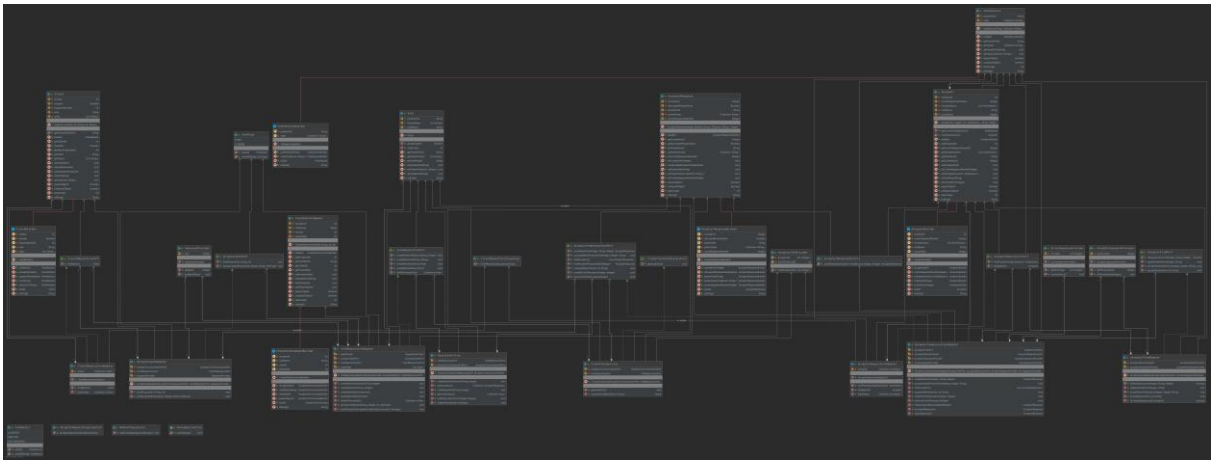
- Java - współbieżny, oparty na klasach, obiektowy język programowania ogólnego zastosowania. Został stworzony przez grupę roboczą pod kierunkiem Jamesa Goslinga z firmy Sun Microsystems. Java jest językiem tworzenia programów źródłowych kompilowanych do kodu bajtowego, czyli postaci wykonywanej przez maszynę wirtualną. Język cechuje się silnym typowaniem. Jego podstawowe koncepcje zostały przejęte z języka Smalltalk (maszyna wirtualna, zarządzanie pamięcią) oraz z języka C++ (duża część składni i słów kluczowych).
- Spring Framework – szkielet tworzenia aplikacji (ang. application framework) w języku Java dla platformy Java Platform, Enterprise Edition. Spring Framework powstał jako alternatywa dla programowania aplikacji z użyciem Enterprise JavaBeans. Programowanie z użyciem EJB narzucało wiele ograniczeń – wymagając między innymi przyjęcia określonego modelu tworzenia oprogramowania.Spring Framework oferuje dużą swobodę w tworzeniu rozwiązań, a jednocześnie jest dobrze udokumentowany i zawiera rozwiązania wielu zagadnień, często występujących w programowaniu. Podczas gdy bazowe komponenty Springa mogą być używane praktycznie w każdej aplikacji, istnieje w nim wiele rozszerzeń, które pozwalają budować aplikacje webowe na bazie Java EE.

Frontend:

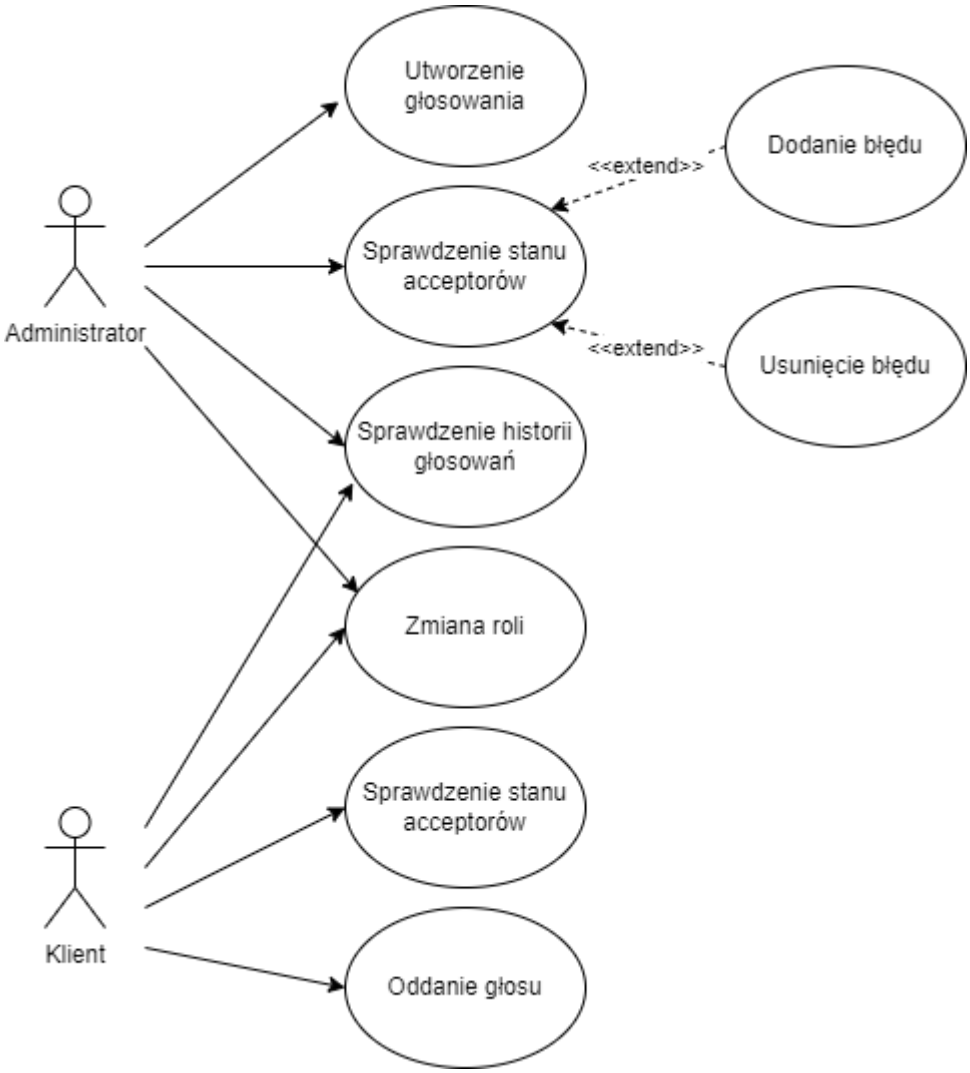
- Angular – otwarty framework i platforma do tworzenia SPA (single page application), napisany w języku TypeScript i wspierany oraz rozwijany przez Google. Angular początkowo miał być wersją 2 popularnego frameworku AngularJS, jednak decyzje projektowe sprawiły, że Google zdecydował się wydać go jako osobny byt, m.in. ze względu na brak kompatybilności wstecznej oraz prostej ścieżki aktualizacji aplikacji napisanych w AngularJS do Angular 2. Angular wydany jest na licencji MIT

3. Opis implementacji.

3.1.Diagram klas:



3.2. Diagram przypadków użycia:



3.3. Backend:

Główną klasą z której wywoływane są wszystkie metody jest PaxosController.java:

```java
@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
public class PaxosController {

    private final VoteRepositoryPort voteRepositoryPort;
    private final AcceptorCommunicationPort acceptorCommunicationPort;

    @PostMapping("/create-new-vote-session")
    public void createNewVoteSession(@RequestParam String voteName, @RequestParam Integer
clientId) {
        voteRepositoryPort.createNewVoteSession(voteName, clientId);
    }

    @GetMapping("/acceptors")
    public Collection<AcceptorResponse> getAcceptors() {
        return acceptorCommunicationPort.findAcceptors();
    }

    @PostMapping("/create-vote")
    public void createNewVote(@RequestParam String vote, @RequestParam Integer clientId) {
        voteRepositoryPort.createNewVote(vote, clientId);
    }

    @GetMapping("/votes")
    public Collection<Vote> getVoteHistory() {
        return voteRepositoryPort.findAllHistoryVote();
    }

    @PostMapping("/create-error")
    public void enableAcceptorError(@RequestParam Integer acceptorId, @RequestParam Integer
errorType) {
        acceptorCommunicationPort.createErrorOnAcceptor(acceptorId, errorType);
    }

    @PostMapping("/delete-error")
    public void disableAcceptorError(@RequestParam Integer acceptorId) {
        acceptorCommunicationPort.removeErrorOnAcceptor(acceptorId);
    }
}
```

Jest to kontroler, w którym znajdują się mapowania API. Funkcje kolejno odpowiadają za:
Tworzenie nowej sesji głosowania,
Pobrania listy akceptorów,
Głosowanie,
Pobrania historii głosowań,
Wprowadzeniu błędu do akceptora,
Usunięcie błędu z akceptora

Najważniejszymi metodami są polecenie utworzenia nowej sesji głosowania i polecenie głosowania, utworzenie sesji wywołuje się metodą z Service'u VoteRepositoryAdapter – createNewVoteSession()

```
@Service
@RequiredArgsConstructor
public class VoteRepositoryAdapter implements VoteRepositoryPort {
    private final SequenceProvider seqProvider;
    private final AcceptorGatePort acceptorGatePort;
    private final ClientRepositoryPort clientRepositoryPort;
    private final List<Vote> votesData = new LinkedList<>();

    @Override
    public void createNewVoteSession(String voteName, Integer clientId) {
        final int presentSeq = seqProvider.getSeq();
        seqProvider.seqNextValue();
        activateVoteNodes(voteName, clientId, presentSeq, VoteType.START);
    }

    @Override
    public void createNewVote(String voteName, Integer clientId) {
        int clientSeq = clientRepositoryPort.findById(clientId).getSequenceNumber();
        activateVoteNodes(voteName, clientId, clientSeq, VoteType.CURRENT);
    }

    @Override
    public void createNewVoteHistory(Vote vote) {
        votesData.add(vote);
    }

    @Override
    public void updateByVoteName(Vote vote) {
      votesData.remove(vote);
      votesData.add(vote);
    }

    @Override
    public Collection<Vote> findAllHistoryVote() {
        return votesData;
    }

    private void activateVoteNodes(String voteName, Integer clientId, int seq, VoteType
voteType) {
        AcceptorIdsProvider.findAcceptorIds()
                .forEach(acceptorId -> {
                    CreateVoteCommand command = CreateVoteCommand.builder()
                            .acceptorId(acceptorId)
                            .clientId(clientId)
                            .presentSeq(seq)
                            .voteName(voteName)
                            .build();
                    sendProposeOrAccepted(command,voteType);
                });
    }

    private void sendProposeOrAccepted(CreateVoteCommand command,VoteType voteType) {
        acceptorGatePort.sendPropose(command.getAcceptorId(), command.getVoteName(),
command.getPresentSeq());
        acceptorGatePort.sendAcceptedPropose(command.getAcceptorId(), command.getClientId(),
command.getVoteName(),voteType);
    }
}
```

Funkcja ta dla każdego akceptora wywołuje polecenia sendPropose i sendAcceptedPropose dostawcy usług AcceptorGateAdapter

```
@Service
@RequiredArgsConstructor
public class AcceptorGateAdapter implements AcceptorGatePort {
    private final AcceptorCommunicationPort acceptorCommunicationPort;
    private final ClientRepositoryPort clientRepositoryPort;
    private final SequenceProvider sequenceProvider;

    @Override
    public void sendPropose(int acceptorId, String voteName, int currentSeq) {
        acceptorCommunicationPort.createNewVote(acceptorId, voteName, currentSeq);
    }

    @Override
    public void sendAcceptedPropose(int acceptorId, Integer clientId, String voteName,
VoteType voteType) {
        Client presentClient = clientRepositoryPort.findById(clientId);
        presentClient.setSequenceNumber(sequenceProvider.getSeq());
        if (Objects.equals(voteType, VoteType.START))
            acceptorCommunicationPort.acceptedNewProposeVote(acceptorId,
presentClient.getSequenceNumber(), voteName);
        else
            acceptorCommunicationPort.acceptedNewVote(acceptorId,
presentClient.getSequenceNumber(), voteName);
    }
}
```

Te dwie funkcje to faza 1B i 2A przedstawione we wstępie.

Funkcja sendPropose wywołuje funkcję createNewVote z Service'u AcceptorCommunicationAdapter:

```
@Service
@RequiredArgsConstructor
public class AcceptorCommunicationAdapter implements AcceptorCommunicationPort {
    private final AcceptorFlowPort acceptorFlowPort;
    private final AcceptorRepositoryPort acceptorRepositoryPort;
    private final AcceptorSequenceProvider acceptorSequenceProvider;
    private final ErrorSequenceProvider errorSequenceProvider;

    @Override
    public AcceptorResponse createNewVote(Integer acceptorId, String voteName, Integer seq) {
        if (acceptorFlowPort.isSequenceCorrect(acceptorId, voteName, seq)) {
            acceptorFlowPort.updateVoteName(acceptorId, voteName);
            return acceptedResponse();
        }
        return rejectResponse();
    }

    @Override
    public void acceptedNewProposeVote(Integer acceptorId, Integer newSeq, String
acceptedValue) {
        acceptorFlowPort.acceptNewVoteSession(acceptorId, newSeq, acceptedValue);
    }

    @Override
    public List<AcceptorResponse> findAcceptors() {
        return AcceptorIdsProvider.findAcceptorIds()
                .stream()
                .map(this::findAcceptorResponseById)
                .collect(Collectors.toList());
    }

    @Override
    public void acceptedNewVote(int acceptorId, int sequenceNumber, String voteName) {
        acceptorFlowPort.acceptNewVote(acceptorId, sequenceNumber, voteName);
    }

    @Override
    public void createErrorOnAcceptor(Integer acceptorId, Integer errorType) {
        Acceptor acceptor = acceptorRepositoryPort.findById(acceptorId);
        acceptor.setCurrentError(Integer.valueOf(errorType));
    }

    @Override
    public void removeErrorOnAcceptor(Integer acceptorId) {
        errorSequenceProvider.removeErrors();
        Acceptor acceptor = acceptorRepositoryPort.findById(acceptorId);
```

```
        acceptor.setCurrentError(0);
    }

    @Override
    public AcceptorResponse findAcceptorResponseById(Integer acceptorId) {
        Acceptor acceptor = acceptorRepositoryPort.findById(acceptorId);
        acceptorSequenceProvider.setPreviousSeq(acceptor.getCurrentSequenceNumber());
        if (acceptor.getCurrentError() == 1) {
            errorSequenceProvider.getErrorSeq().add(acceptor.getCurrentSequenceNumber());
            if (errorSequenceProvider.getErrorSeq().size() > 1) {
                errorSequenceProvider.getErrorSeq().remove(1);
            }
        }

        VoteSession presentVotingSession =
acceptorRepositoryPort.findPresentVotingSession(acceptorId);
        Integer firstError = acceptor.getCurrentError() == 1  ?
errorSequenceProvider.getErrorSeq().get(0) : null;
        return AcceptorResponseFactory.buildResponse(acceptor, presentVotingSession,
acceptorSequenceProvider.getPreviousSeq(), firstError);
    }

    private AcceptorResponse acceptedResponse() {
        return AcceptorResponse.builder().isAcceptedProposeVote(true).build();
    }

    private AcceptorResponse rejectResponse() {
        return AcceptorResponse.builder().isAcceptedProposeVote(false).build();
    }
}
```

Funkcja createNewVote natomiast sprawdza, czy numer sekwencji jest poprawny, jeżeli tak, aktualizowana jest nazwa (przedmiot głosowania) i zwracana jest odpowiedź pozytywna, w przeciwnym wypadku propozycja jest odrzucana przez akceptora. Następnie wywoływana jest usługa acceptNewVoteSession z AcceptorFlowAdapter

```
@Service
@RequiredArgsConstructor
public class AcceptorFlowAdapter implements AcceptorFlowPort {
    private final AcceptorRepositoryPort acceptorRepositoryPort;
    private final AcceptorSequenceProvider acceptorSequenceProvider;

    @Override
    public boolean isSequenceCorrect(Integer acceptorId, String voteName, Integer seq) {

acceptorSequenceProvider.setPreviousSeq(acceptorRepositoryPort.findById(acceptorId).getCurrent
SequenceNumber());
        return
acceptorRepositoryPort.findById(acceptorId).getCurrentSequenceNumber().equals(seq);
    }

    @Override
    public void updateVoteName(Integer acceptorId, String voteName) {
        acceptorRepositoryPort.findById(acceptorId).setVoteName(voteName);
    }

    @Override
    public void acceptNewVoteSession(Integer acceptorId, Integer newSeq, String acceptedValue)
{
        Acceptor acceptor = acceptorRepositoryPort.findById(acceptorId);
        List<VoteSession> currentVoteSession = acceptor.getVotingSessions();

currentVoteSession.add(VoteSession.builder().presentVote(acceptedValue).votes(Collections.empt
yList()).build());
        acceptor.setVotingSessions(currentVoteSession);
        acceptor.setCurrentSequenceNumber(newSeq);
    }

    @Override
    public void acceptNewVote(int acceptorId, int sequenceNumber, String voteName) {
        Acceptor acceptor = acceptorRepositoryPort.findById(acceptorId);
        Collection<String> votes = new
LinkedList<>(acceptor.getCurrentVotingSession().getVotes());
        votes.add(voteName);

        if (isCurrentSequence(sequenceNumber, acceptor)) {
            acceptor.getCurrentVotingSession().setVotes(votes);
```

```
            acceptor.setCurrentSequenceNumber(sequenceNumber);
        }
    }

    private boolean isCurrentSequence(int currentSeq, Acceptor acceptor) {
        return acceptor.getCurrentSequenceNumber().equals(currentSeq) &&
acceptor.getCurrentError() == 0;
    }
}
```

Dana sesja ustawiana jest jako aktualna.

Funkcja z kontrolera createNewVote przechodzi przez podobne funkcje, z tą różnicą, że jako ostatnia funkcja wywoływana jest acceptNewVote zamiast acceptNewVoteSession, w której sprawdzany jest numer sekwencji danego głosowania i acceptora.

Zaimplementowane błędy:

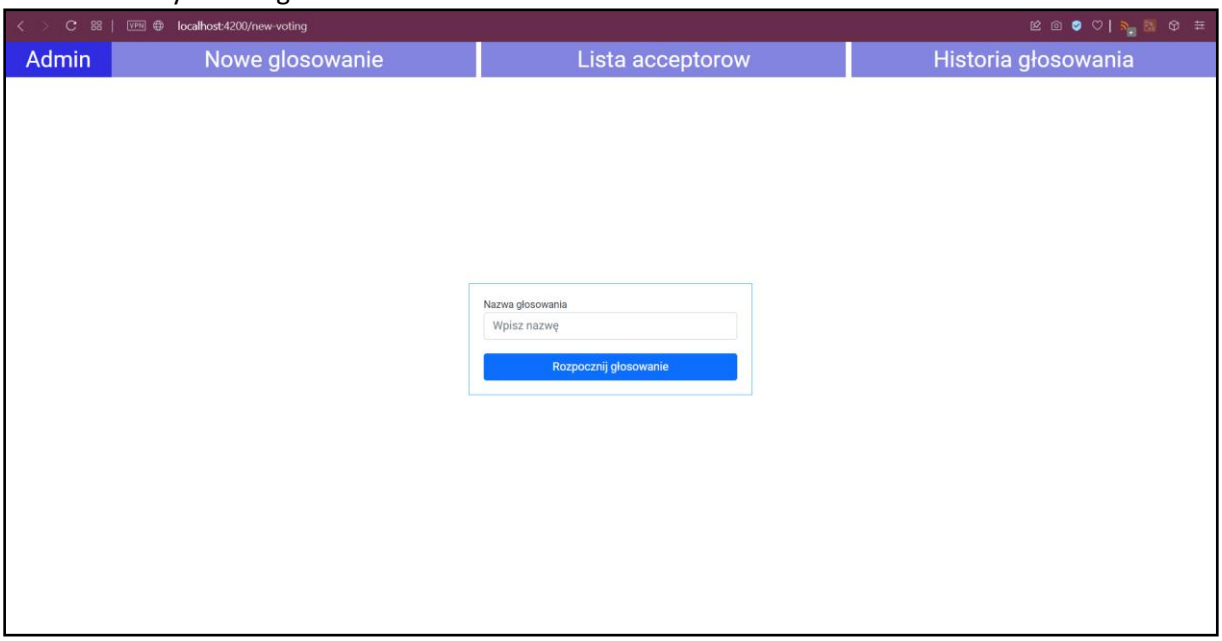
- W acceptorze może się zepsuć inkrementacja sekwencji
- Sekwencja w acceptorze może być ujemna

3.4. Interfejs graficzny

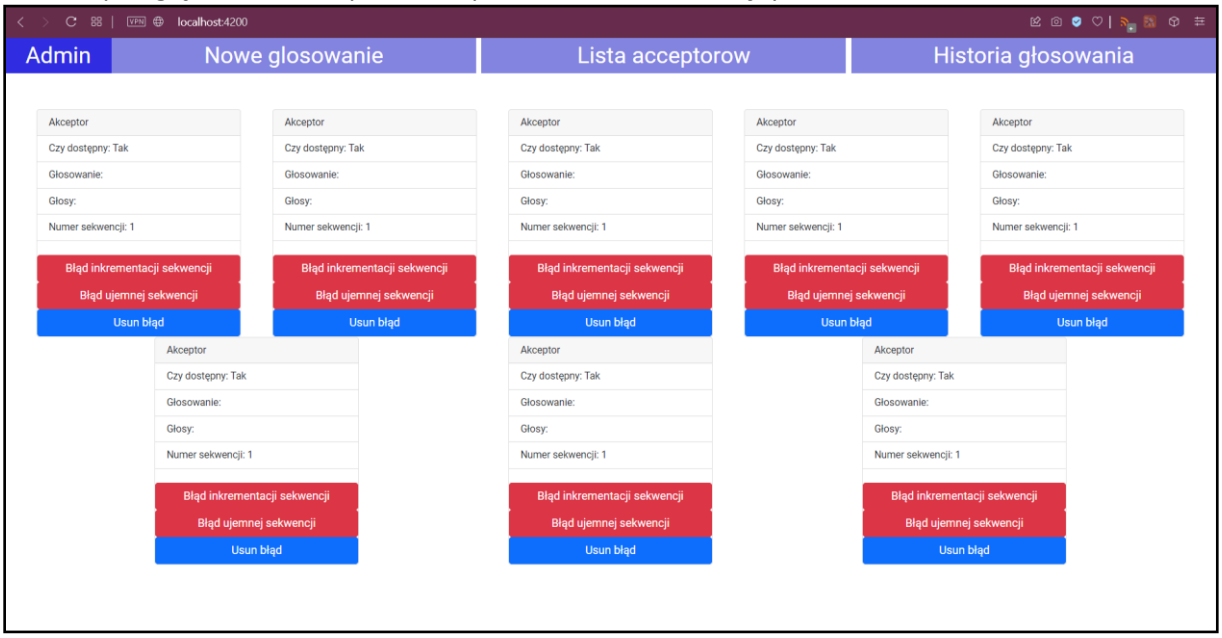
Interfejs podzielony jest na panel administratora i panel klienta.

Administrator może:

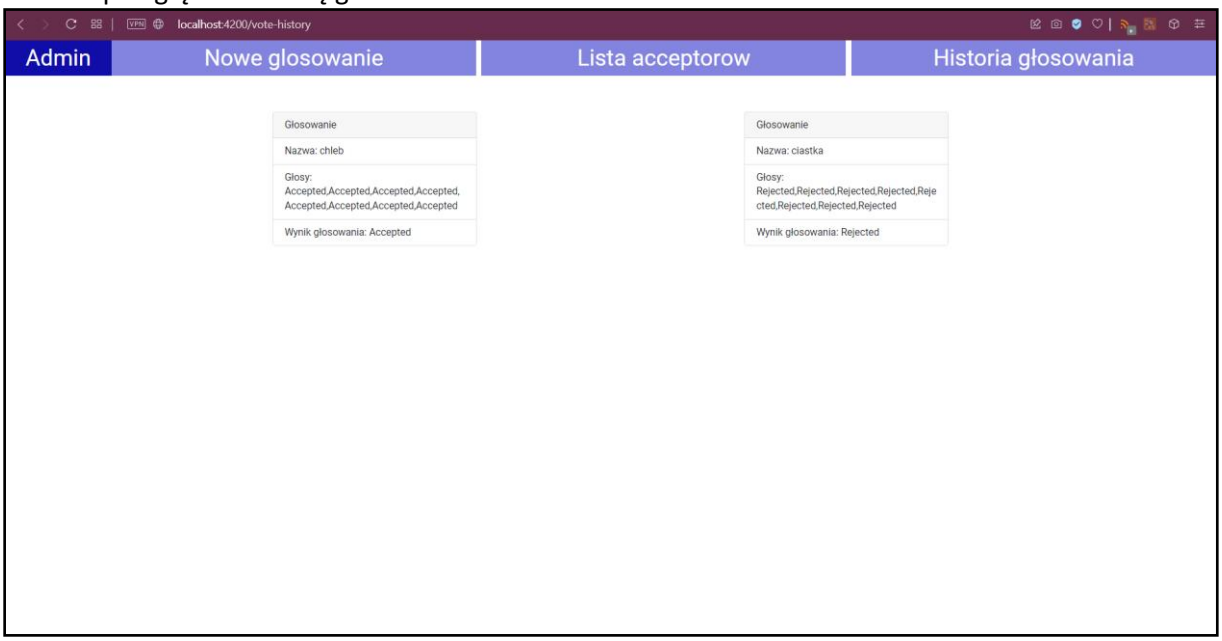
- tworzyć nowe głosowanie:



- przeglądać stan acceptorów, wprowadzać i usuwać błędy

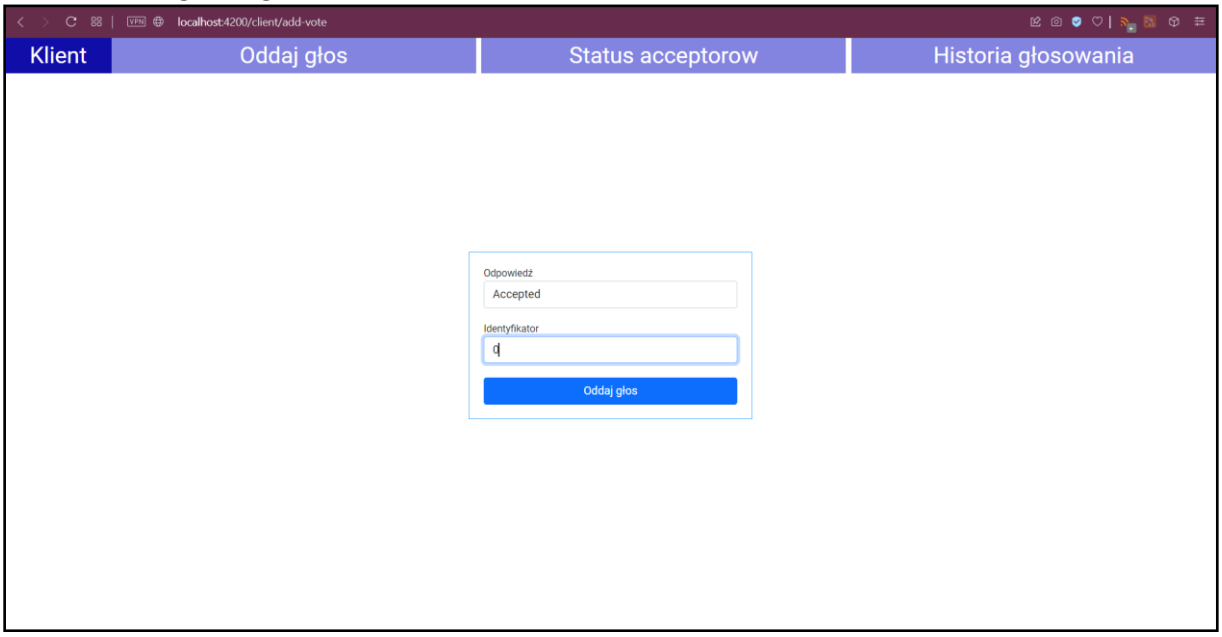


- przeglądać historię głosowania

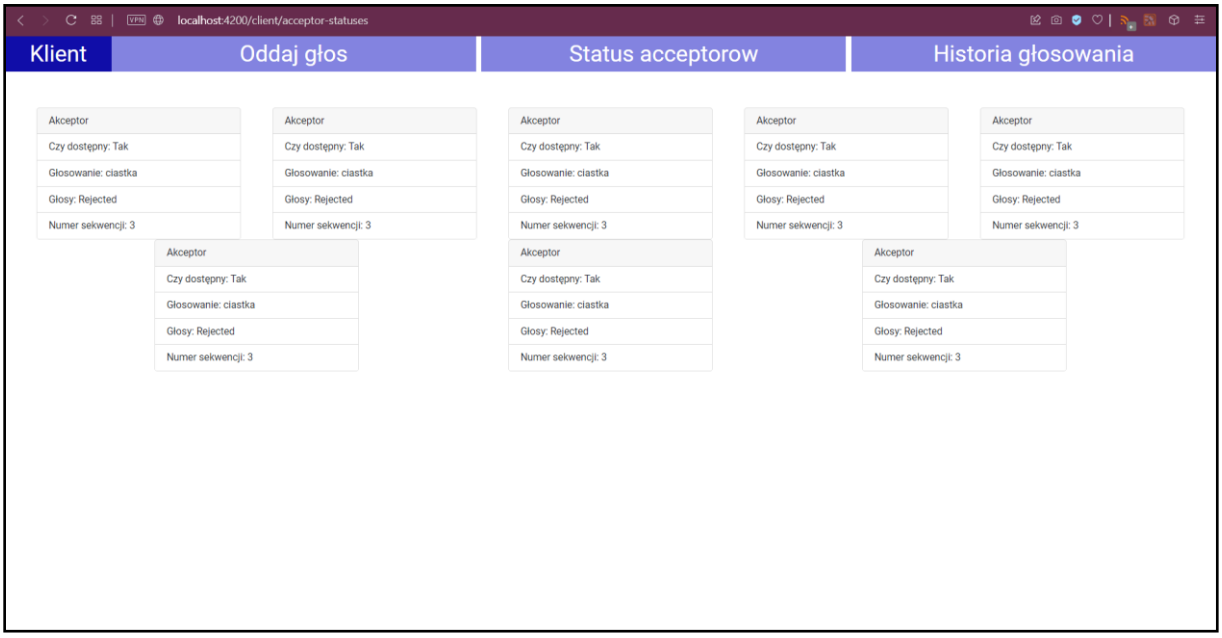


Klient może:

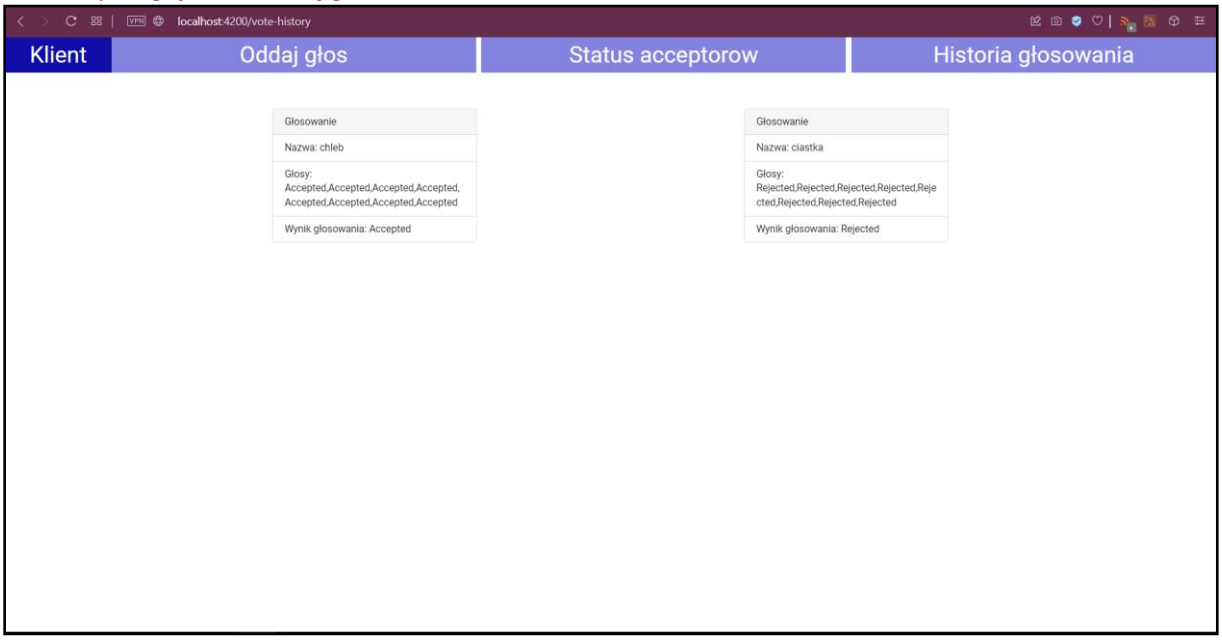
- oddać głos w głosowaniu,



- przeglądać stan acceptorów

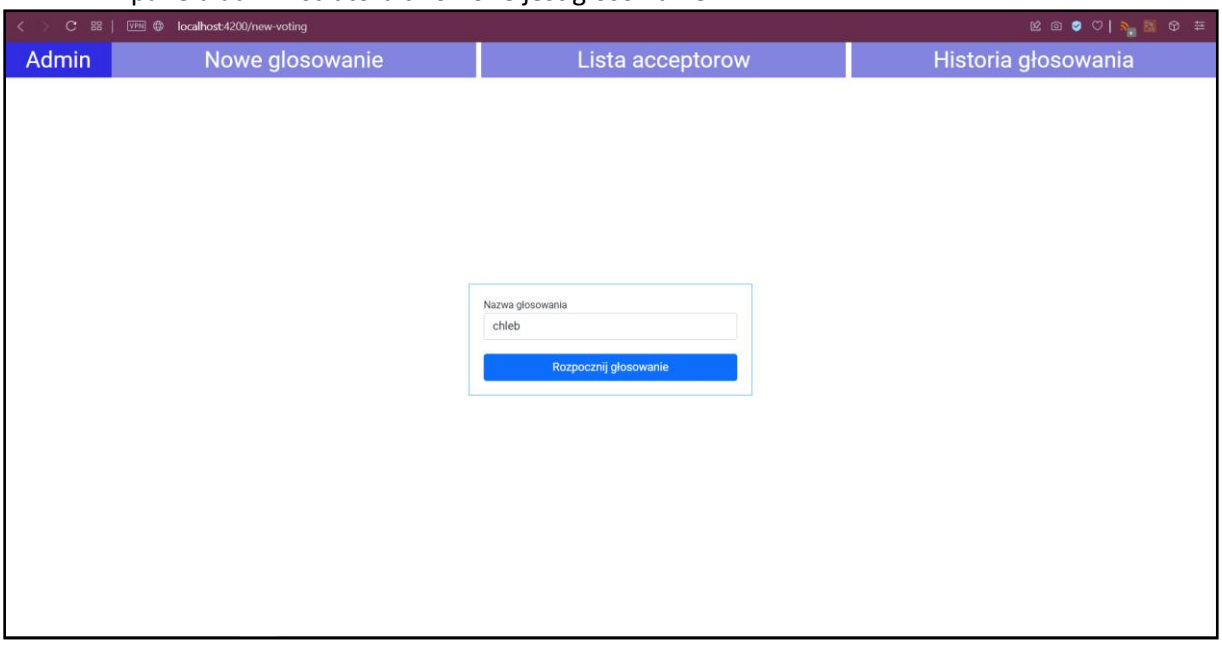


- przeglądać historię głosowania

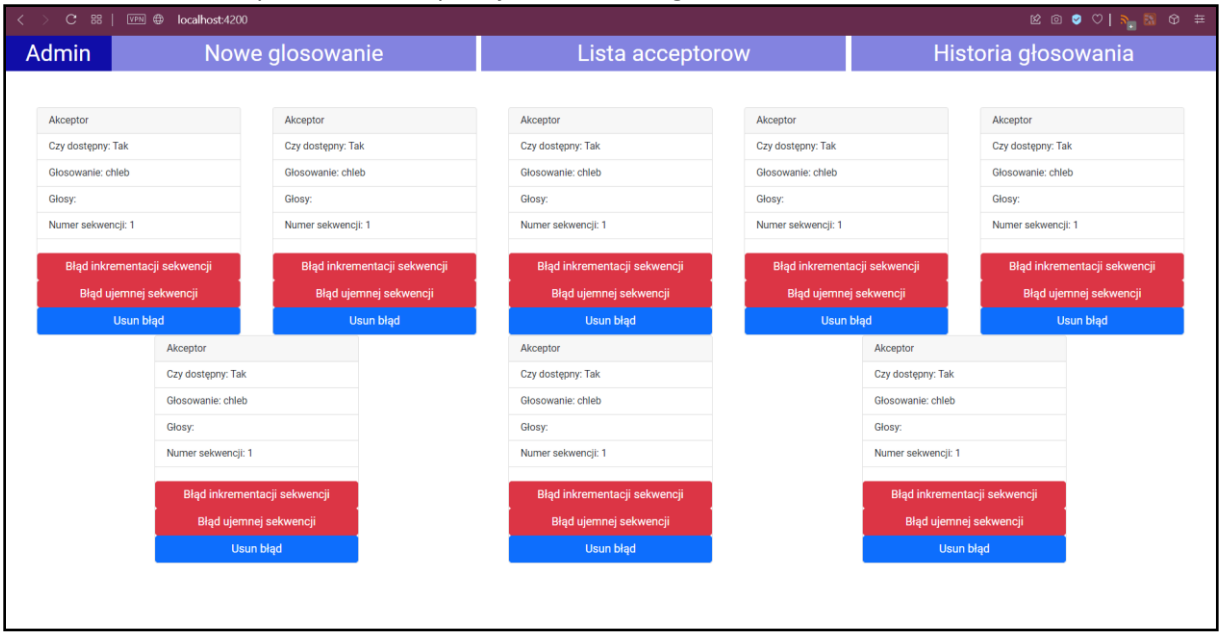


4. Przykład głosowania:

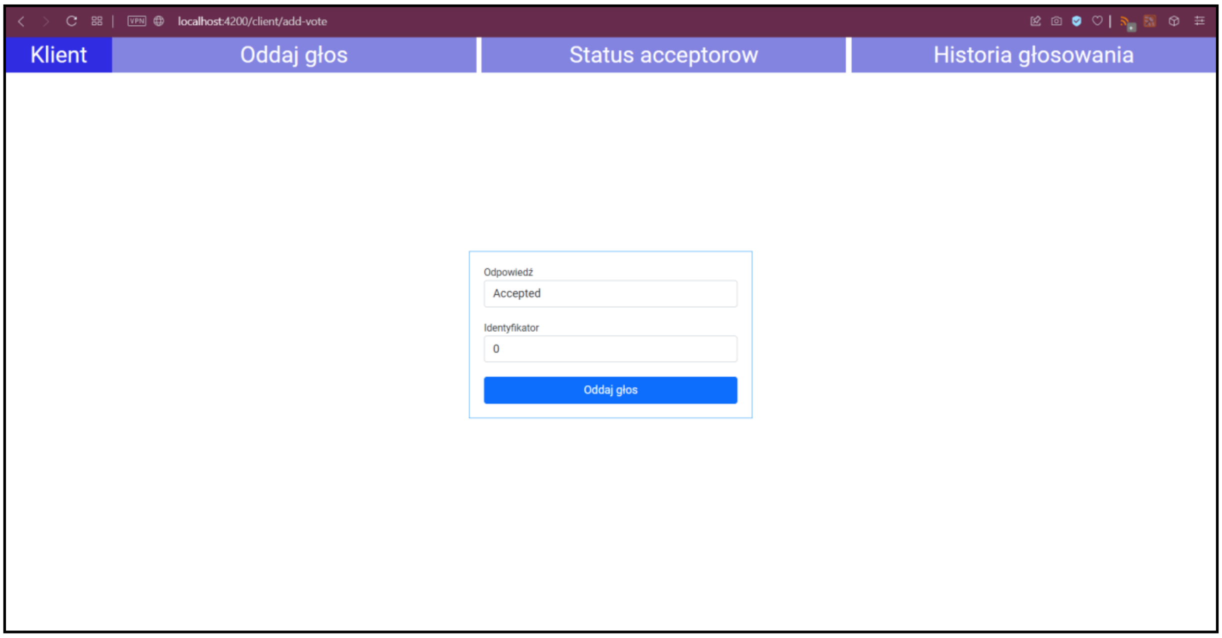
1. W panelu administratora tworzone jest głosowanie.



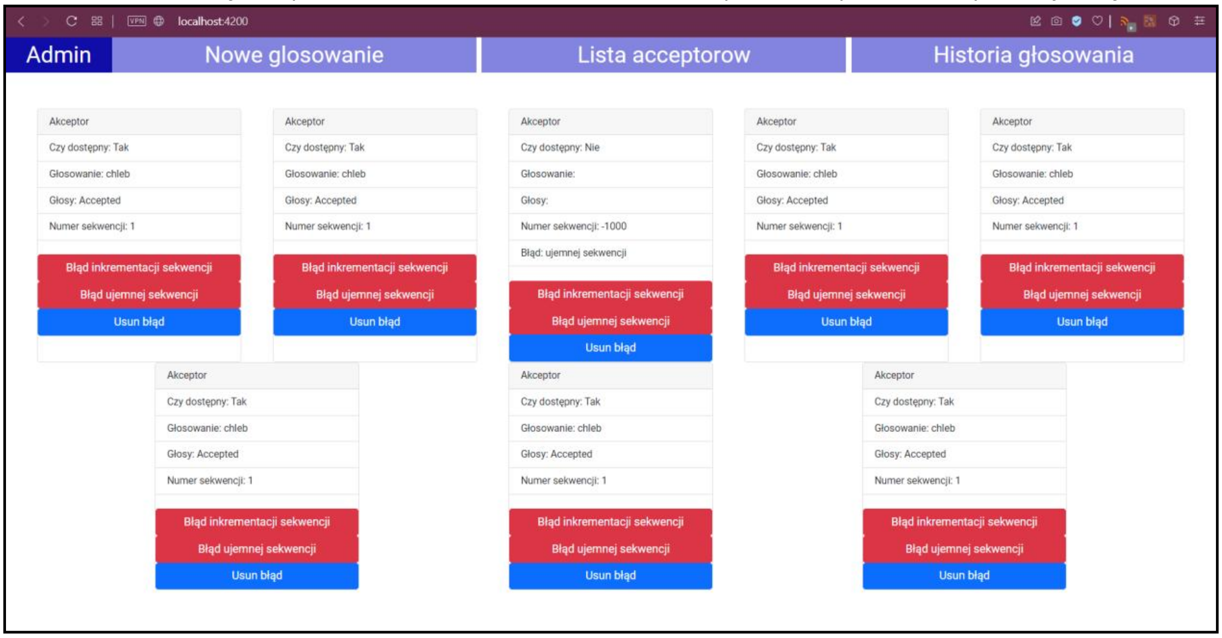
2. Na liście acceptorów można podejrzeć aktualne głosowanie.



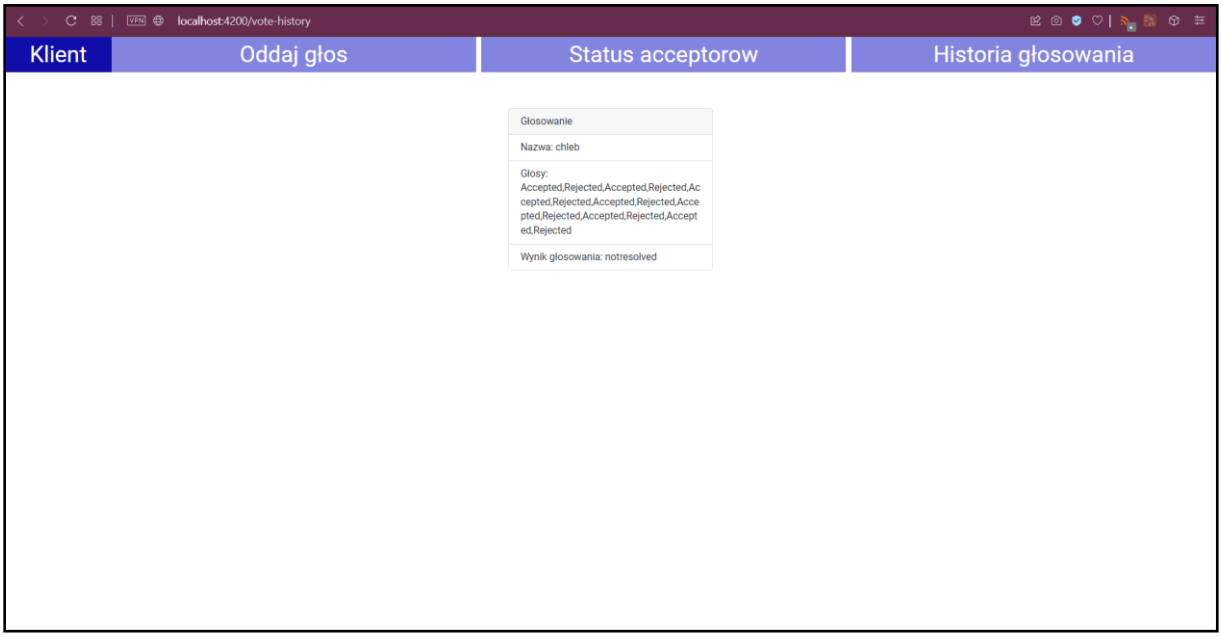
3. W panelu klienta można oddać głos, wybierana jest wartość z listy Accepted lub Rejected i podawany jest identyfikator klienta.



4. Przechodząc do panelu administratora i statusu acceptorów wprowadzany zostaje błąd.



5. Kolejny klient oddaje głos, sprawdzana jest historia.



Jak widać błąd akceptora nie spowodował błędnego wyniku, ilość głosów na tak i na nie, jest równa. Zakończyliśmy głosowanie bez rozstrzygnięcia.

5. Podsumowanie

Program został wykonany przy połączeniu frontendu w postaci Angulara i backendu przy wykorzystaniu Java i Spring Framework, komunikacja odbywa się przy pomocy metod http GET i POST.

System paxos może zostać wykorzystany do ustalania wartości w procesie głosowania, w przypadku wystąpienia błędów acceptorów wynik dalej jest poprawny, co ukazuje ostatni przykład.

6. Bibliografia:

https://en.wikipedia.org/wiki/Paxos_(computer_science) (dostęp 28.10.2021)
https://pl.wikipedia.org/wiki/Java (dostęp 13.12.2021)
https://pl.wikipedia.org/wiki/Spring_Framework (dostęp 13.12.2021)
https://pl.wikipedia.org/wiki/Angular_(framework) (dostęp 13.12.2021)