容錯計算 part.2 報告

- **Server.py**

```
tz = timezone(timedelta(hours=+8))
Tokens = {}
Due = deque()
Voters = {}
Elections = {}
Challenges = {}
Ballots = {}
```

Tokens 存使用者的 token

Due 存 token 的到期時間

Voters 存使用者的 group 和 key

Elections 存 group, choices, end date 和 token

Challenges 存要給使用者加密的 random challenge

Ballots 存哪些使用者投哪些選項

A. Local Server API

RegisterVoter:

```
def RegisterVoter(name_var, group_var, key_var):
    name = name_var.get()
    group = group_var.get()
    key = ast.literal_eval(key_var.get())
    try:
        if name not in Voters.keys():
            Voters[name] = (group, key)
            PopupWin("Register success!")
            return 0
        else:
            PopupWin("Voter Name already exists!")
            return 1
    except:
        PopupWin("Undefined error.")
        return 2
```

UnregisterVoter:

```
def UnregisterVoter(name_var):
    name = name_var.get()
    try:
        if name in Voters.keys():
            del Voters[name]
            PopupWin("Unregister Success!")
            return 0
        else:
            PopupWin("Voter Name does not exist!")
            return 1
    except:
        PopupWin("Undefined error.")
        return 2
```

B. RPC APIs

GetResult:

```
def GetResult(self, request, context):
    '''
    ElectionReuslt.status = 0: Sucess
    ElectionResult.status = 1: Non-existent election
    ElectionResult.status = 2: The election is still ongoing.
                               Election result is not available yet.
    '''
    try:
        elecname = request.name
        timestamp = Timestamp()
        timestamp.FromDatetime(datetime.now().astimezone(tz))

        if elecname not in Elections.keys():
            return vote.ElectionResult(status=1, count=[])
        if Elections[elecname][2].seconds > timestamp.seconds:
            return vote.ElectionResult(status=2, count=[])

        return vote.ElectionResult(status=0, count=Count_Ballot(elecname))
    except Exception as e:
        print("Result query error: " + str(e))
        return vote.ElectionResult(status=3, count=[])
```

PreAuth / Auth:

```
def checkToken():
    while Due:
        if Due[0][0] < datetime.now():
            del Tokens[Due[0][1]]
            Due.popleft()
        else:
            break
```

```
class eVoting(vote_grpc.eVotingServicer):
    def PreAuth(self, request, context):
        name = request.name
        chal = os.urandom(4)
        Challenges[name] = chal
        return vote.Challenge(value=chal)

    def Auth(self, request, context):
        name = request.name.name
        response = request.response.value
        verify_key = VerifyKey(Voters[name][1])
        try:
            assert Challenges[name] == verify_key.verify(response)
            # PopupWin("Pass.")
            b = os.urandom(4)
            Tokens[name] = b
            Due.append([datetime.now()+timedelta(hours=1), name])
            return vote.AuthToken(value=b)
        except:
            PopupWin("Fail.")
            return vote.AuthToken(value=b'\x00\x00')
```

CreateElection:

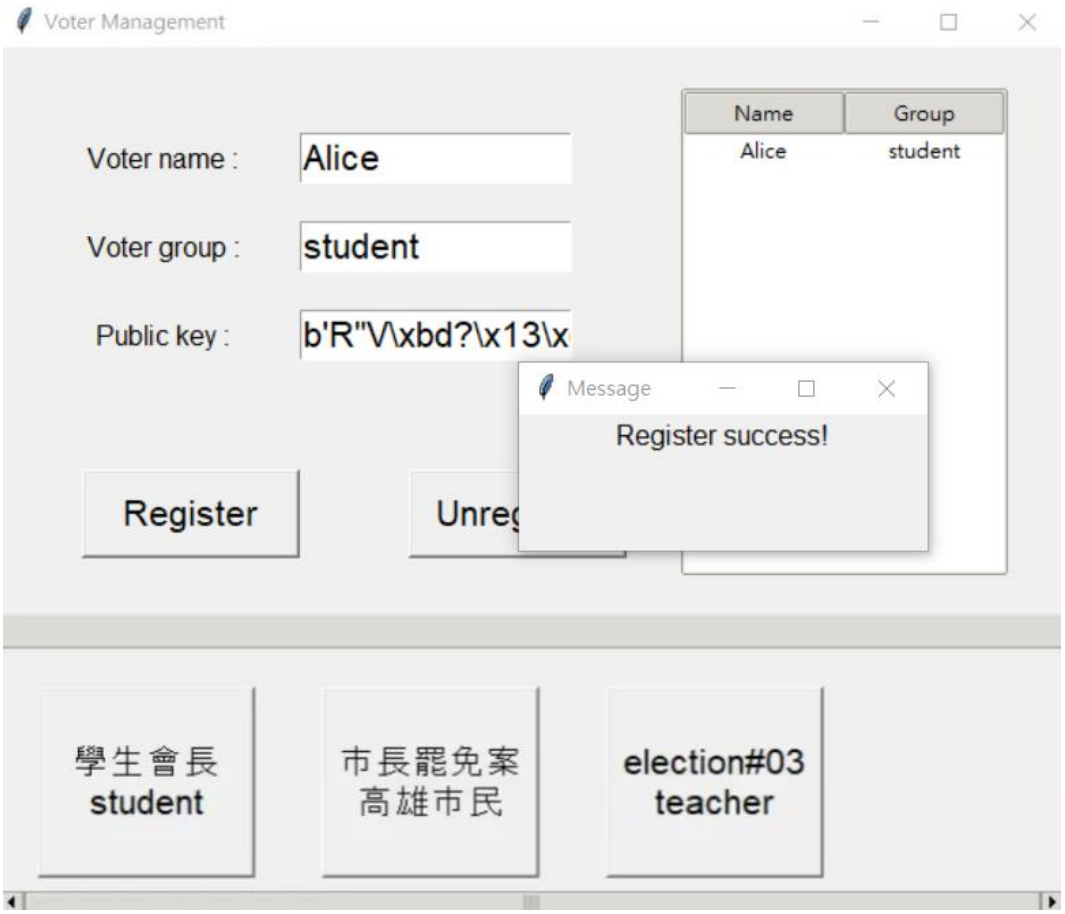
```
def CreateElection(self, request, context):
    checkToken()
    try:
        elecname = request.name
        token = request.token.value
        if token in Tokens.items():
            try:
                groups = request.groups
                choices = request.choices
                end_date = request.end_date
                print("new election : "+elecname+", end at:" +
                      str(datetime.fromtimestamp(end_date.seconds).astimezone(tz)))
                Elections[elecname] = (groups, choices, end_date, token)
                Ballots[elecname] = {}
                #PopupWin("Election created successfully!")
                return vote.Status(code=0)
            except:
                #PopupWin("Missing groups or choices specification!")
                return vote.Status(code=2)
        else:
            #PopupWin("invalid authentication token!")
            return vote.Status(code=1)
    except Exception as e:
        #PopupWin("Undefined error.")
        print("Create Election error: " + str(e))
        return vote.Status(code=3)
```

CastVote:

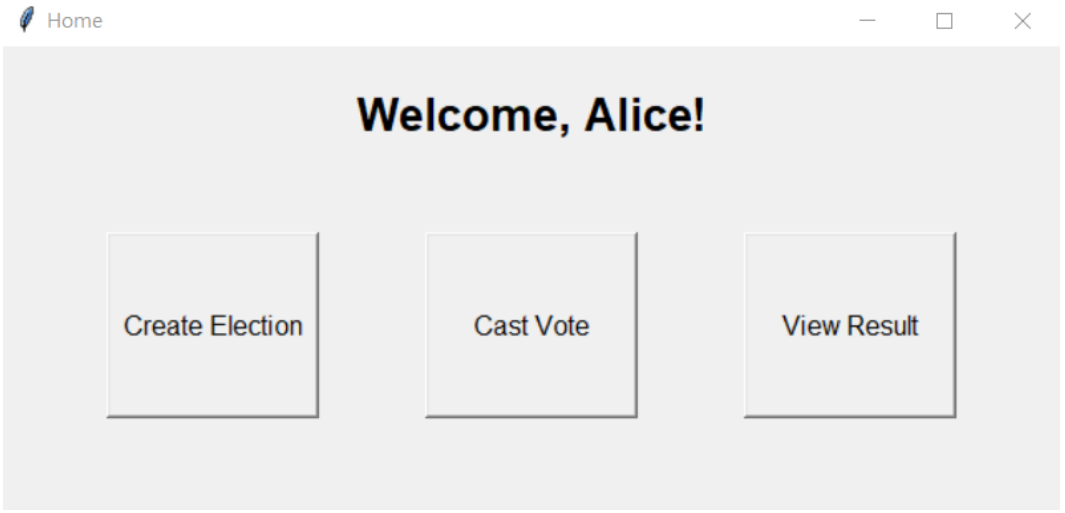
```
def CastVote(self, request, context):
    '''
    Status.code=0 : Successful vote
    Status.code=1 : Invalid authentication token
    Status.code=2 : Invalid election name
    Status.code=3 : The voter's group is not allowed in the election
    Status.code=4 : A previous vote has been cast.
    '''
    checkToken()
    try:
        token = request.token.value
        elecname = request.election_name
        votername = list(Tokens.keys())[list(Tokens.values()).index(token)]
        choice_name = request.choice_name
        print(votername+"->"+elecname+"->"+choice_name)
        if token not in Tokens.values():
            return vote.Status(code=1)
        if elecname not in Elections.keys():
            return vote.Status(code=2)
        if Voters[votername][0] not in Elections[elecname][0]:
            return vote.Status(code=3)
        if votername in Ballots[elecname].keys():
            return vote.Status(code=4)
        if choice_name not in Elections[elecname][1]:  # choice not exist
            return vote.Status(code=5)
        Ballots[elecname][votername] = choice_name
        return vote.Status(code=0)

    except Exception as e:
        print("Cast error: " + str(e))
        return vote.Status(code=5)
```

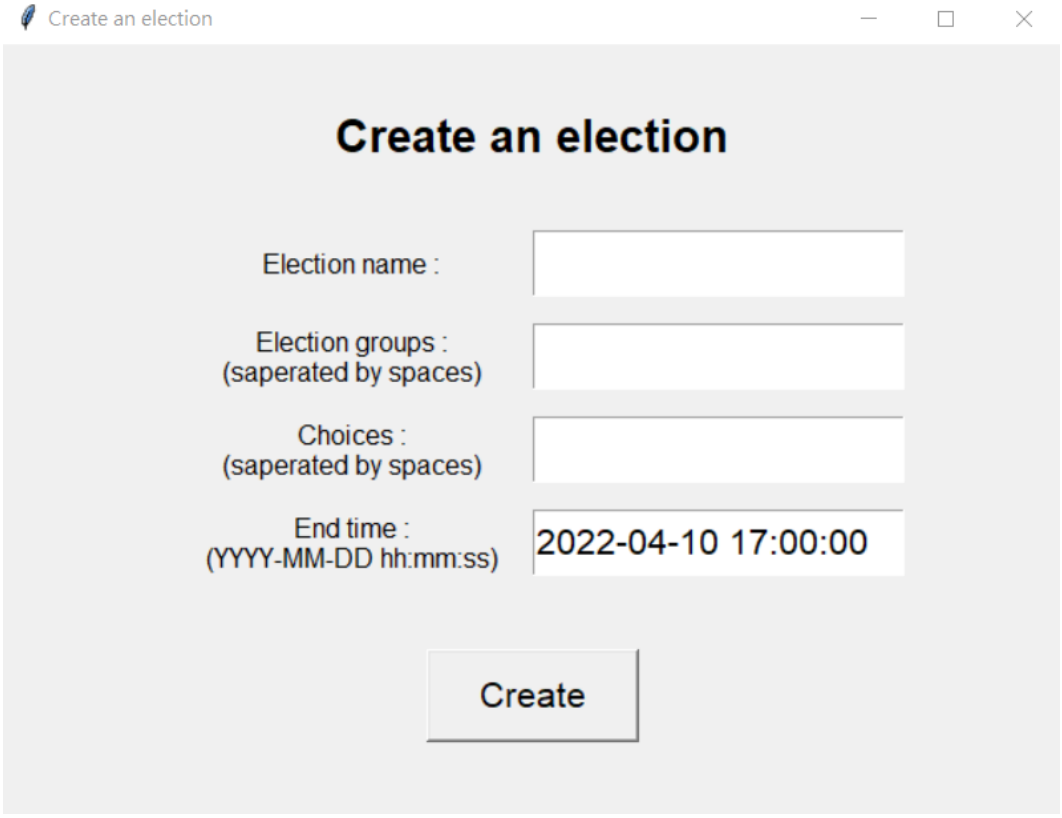
- 執行畫面

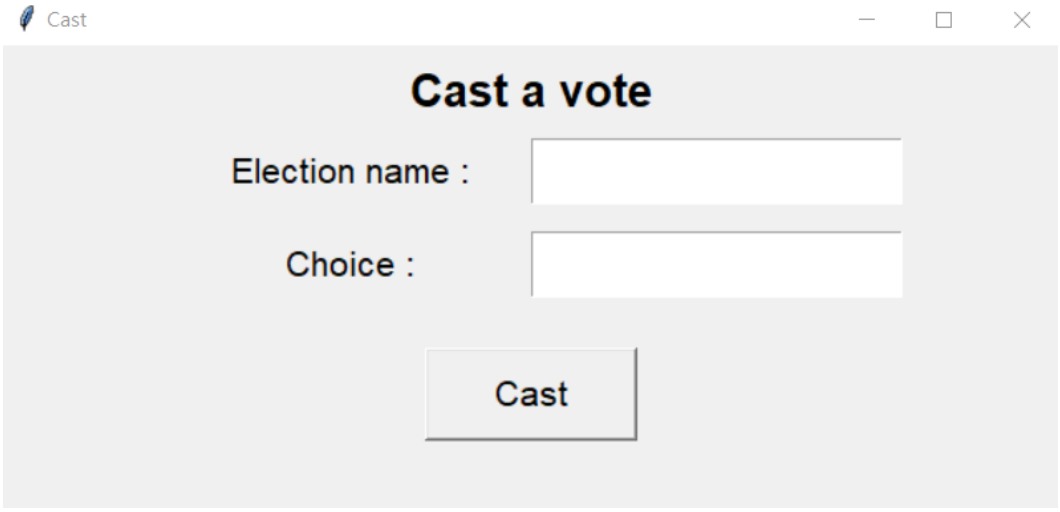
server



client




Create an election
Create an election
Election name :
Election groups :
(saperated by spaces)
Choices :
(saperated by spaces)
End time :
(YYYY-MM-DD hh:mm:ss)
2022-04-10 17:00:00
Create


Cast
Cast a vote
Election name :
Choice :
Cast

Get Result

Get election result

Election name :

Get result

Result