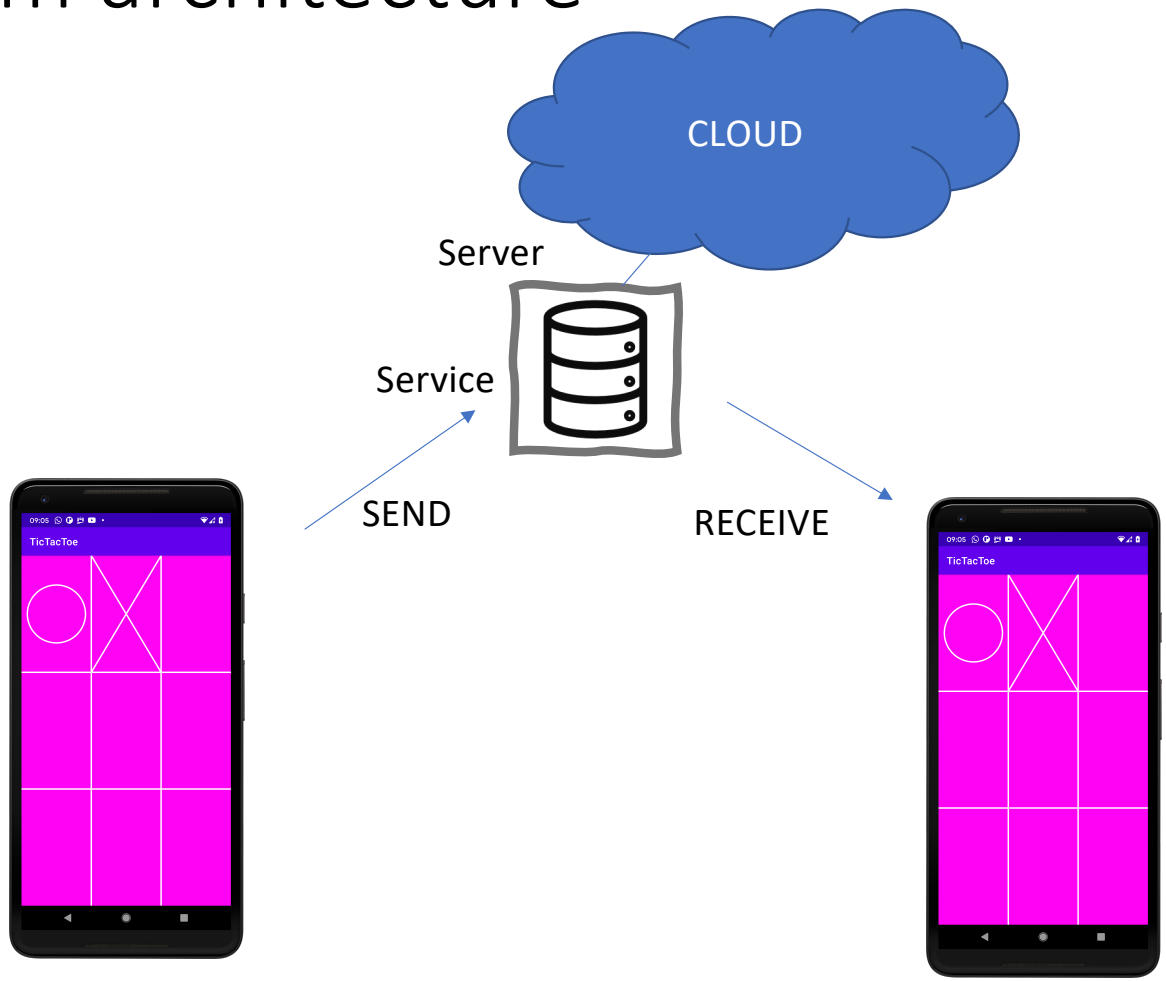
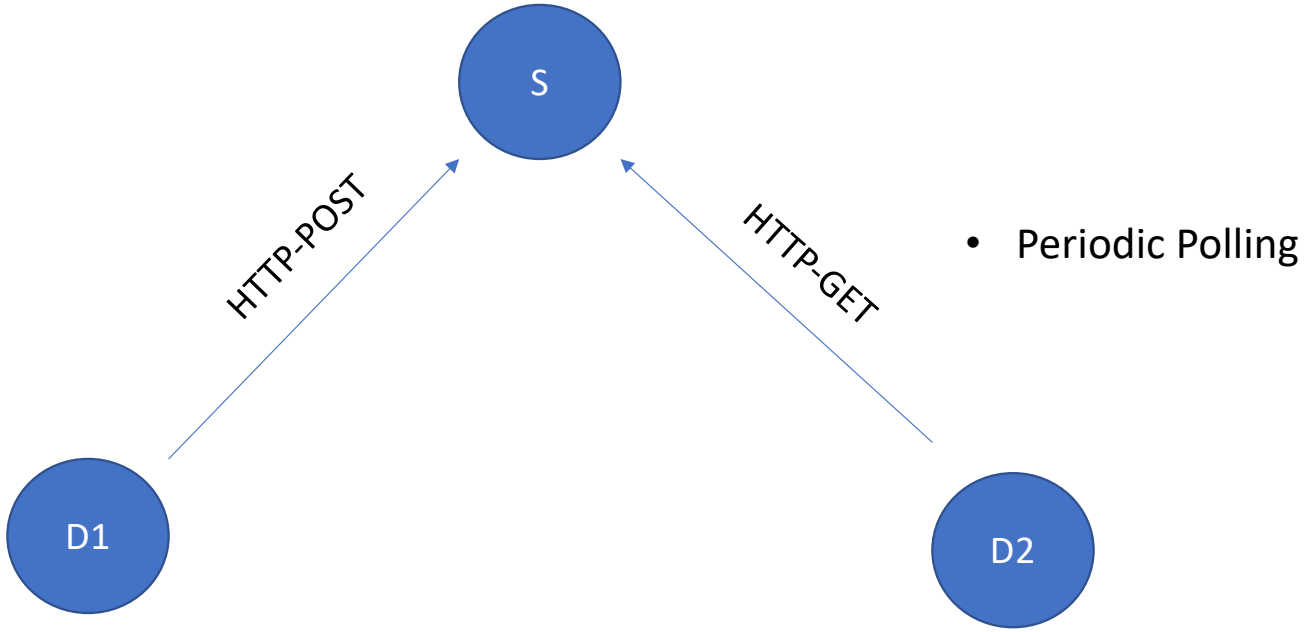
2Players-TICTACTOE

System architecture

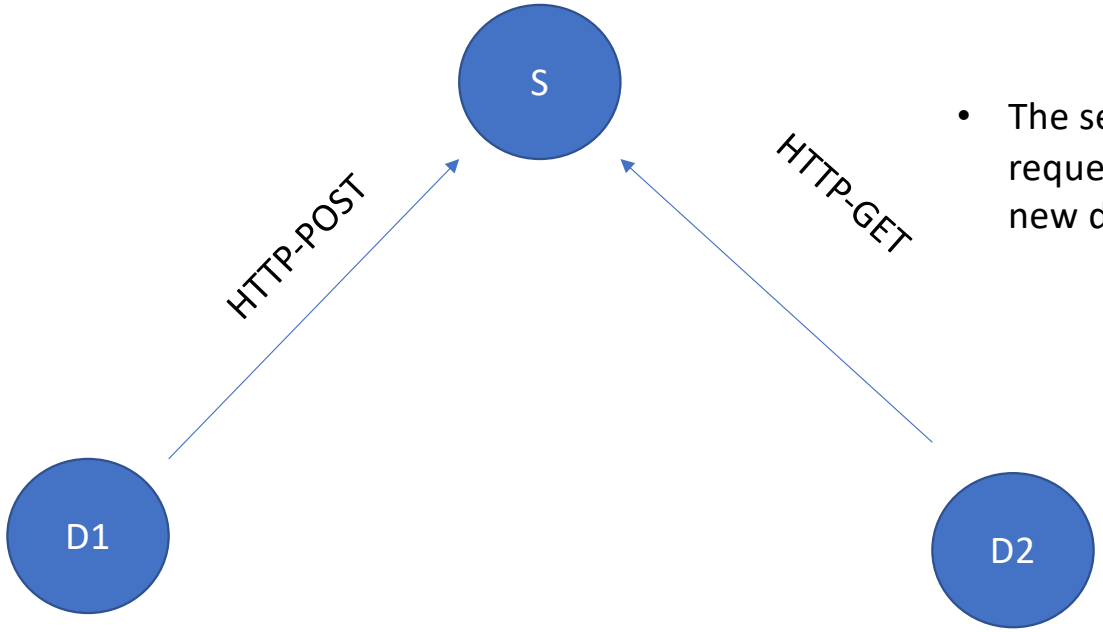


Message over HTTP



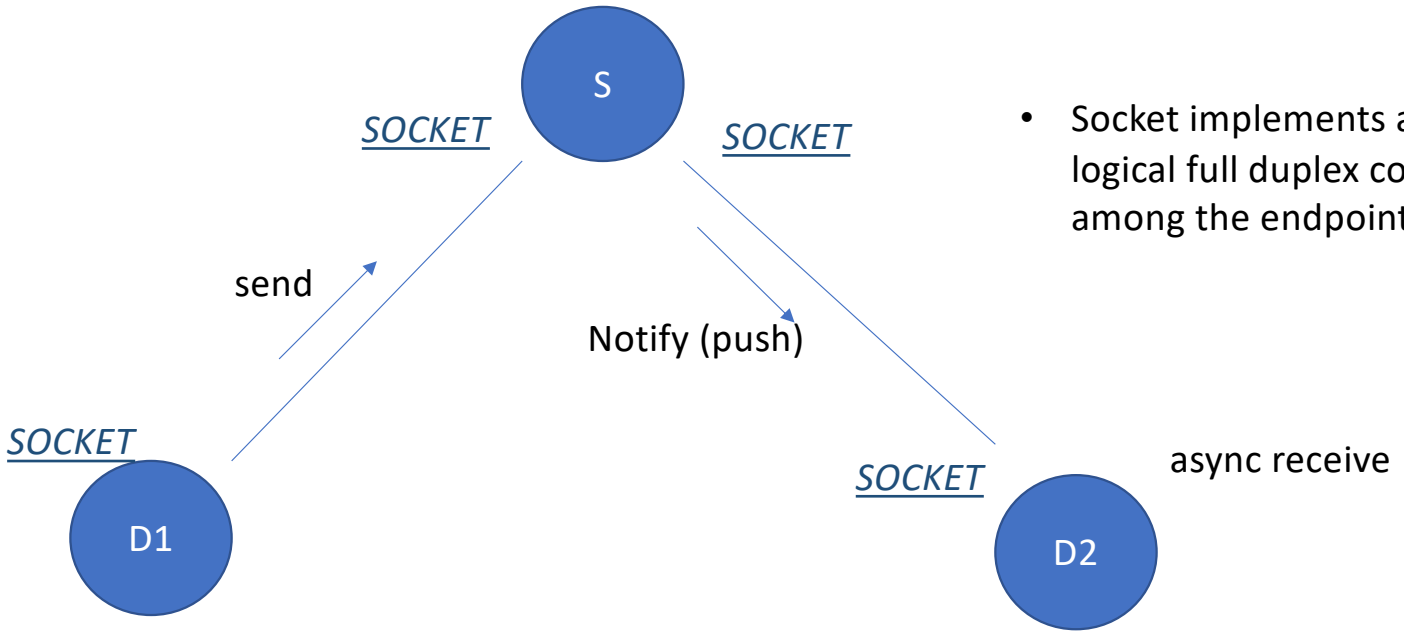
- Periodic Polling

Message over HTTP with long polling



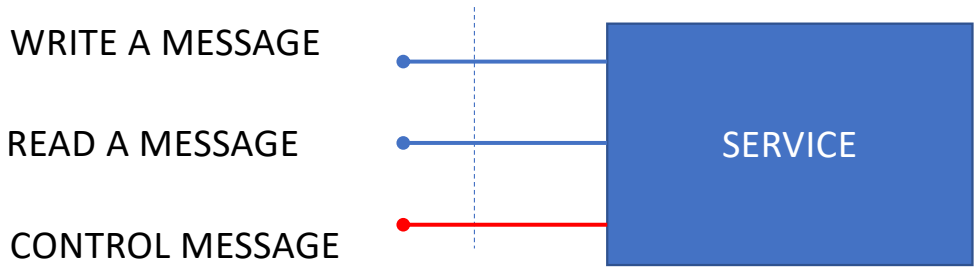
- The server holds the request open until new data is available.

Messaging over WebSockets



- Socket implements a permanent logical full duplex connection among the endpoints

Our solution (version 1)



message type:

RESET(who) → {OK,NOK}
SEND(who, what) → {OK,NOK}
READ(who) → {what}

EXAMPLE

- RESET
- SEND(0,1)
- READ(1)

Implementation strategy in 3 steps: 1/3

- Design, implement one single client (no server)
- Replies are given at random (or computed)
- Challenges:
  - UI
  - Check winner
  - Alternate players

Implementation strategy: 2/3

- Design and implement a messaging service over a web server (suggested: Flask, and Flask-RESTful) on your PC
- Test the server behaviour alone (with curl or postman)
- Modify clients so to interact with the server

Implementation strategy: 3/3

- Move the server on a cloud provider (suggested Pythonanywhere)

- Refectory:
  - Add security aspects (login, secret key, game-key, multiplayer)
  - Test

Design a web-api

- The backend of a mobile app is usually accessed as a Web api
- One may need to design and deploy its own backend service
- To this end we will see some simple example using REST architecture and Python Flask

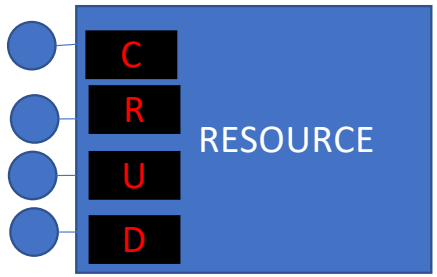
REST: allowed actions on a resource

Create

Read

Update

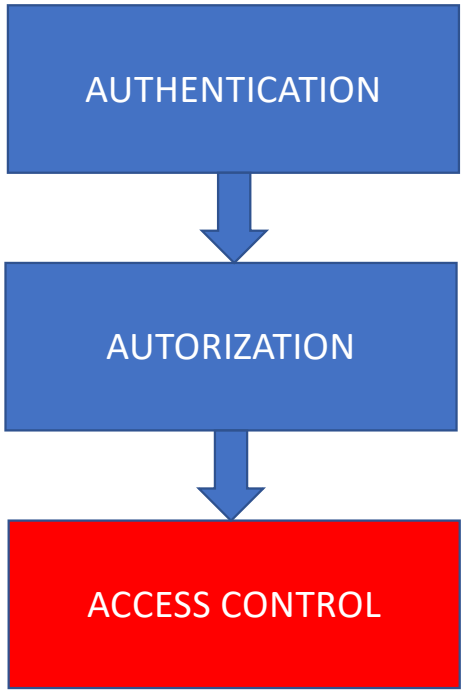
Delete



Managing resources

- **Authentication**: Establish a trusted association between a client and the identity of the principal
- **Authorization**: Determine which resources and operations an authenticated user can perform
- **Access Control**: Check if an operation on a resource is allowed
  - **Access-token:** API call carries a token that has been released after the user has been *authenticated*; check token validity before granting the access
- **Rate control**: How many requests are all allowed?
  - **Throttling:** For example, maximum number of requests per day

Access Control



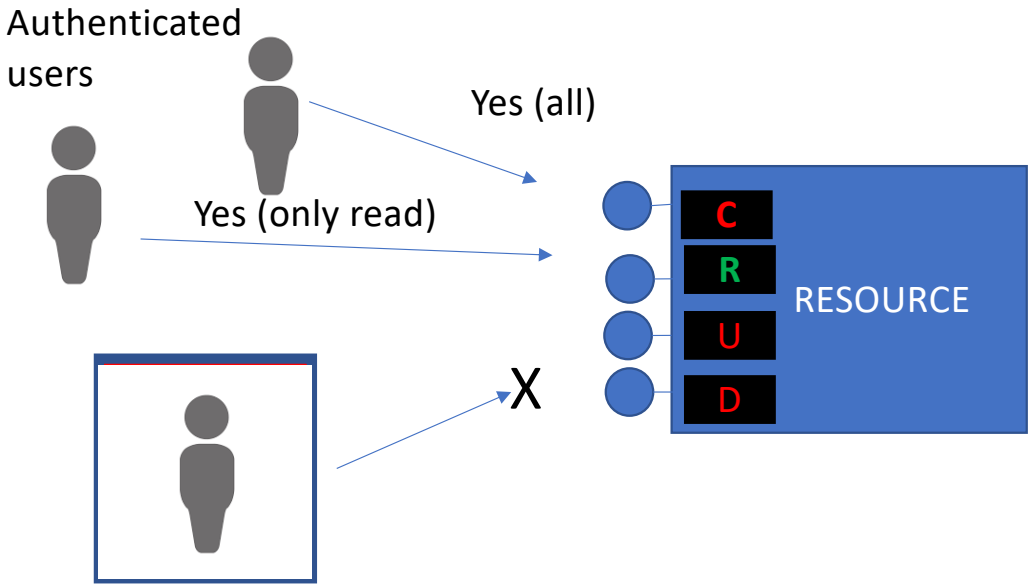
Validate user identity

Assign allowed operations

Control right to perform the op

Token based

Access control

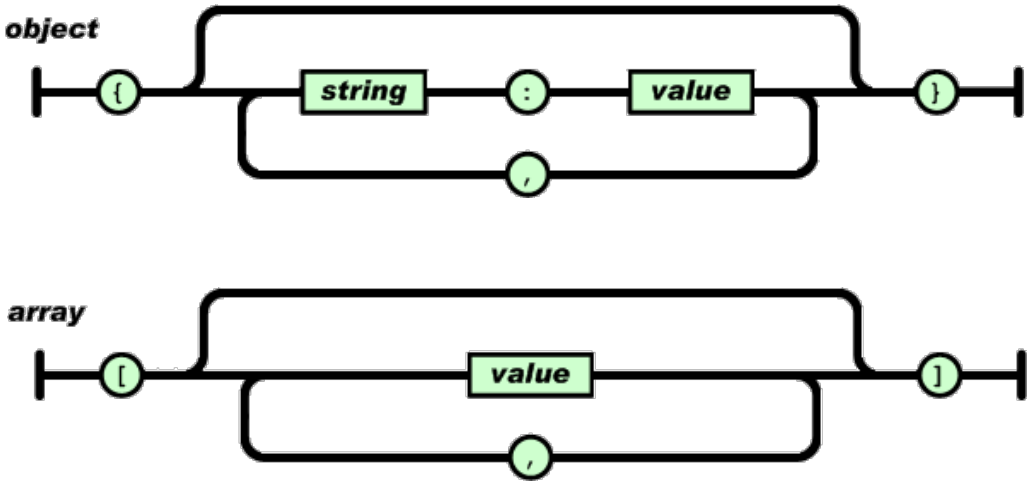


Data Validity check

- Data-type check
  - name@xxx.yyy.zzz
  - Money currency
  - IBAN
- Range check
  - ...<Heart beat < ... Bps
- Consistency check
  - Grandfather > Father > Son
- Presence check

JSON (JavaScript Object Notation)

- **Primitive types**: Number, String, **false**, **true**, **null**
- Two fundamental **structured data**



What about binary data?

JSON serialization/deserialization

- Nowadays all languages have a serializer/de-serializer utility that converts an internal data type into a JSON string and vice versa

A worked example in python flask

- 2 players web-api

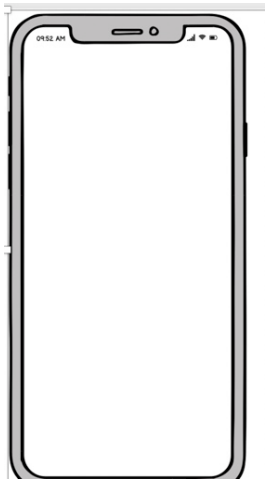
Example of expected behaviour

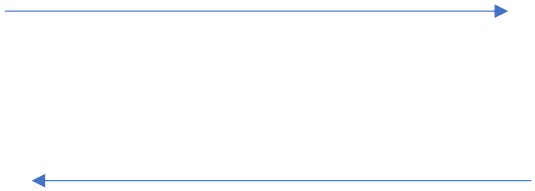
- POLLING(0)-→
- ←LASTMOVE1 (-1)
- NEWGAME→
- ← OK
- MOVE(0,1)→
- ←LASTMOVE0=1
- POLLING(1)→
- ←LASTMOVE0
- POLLING(0)
- ←LASTMOVE1(-1)
- →MOVE(1,5)
- ←OK

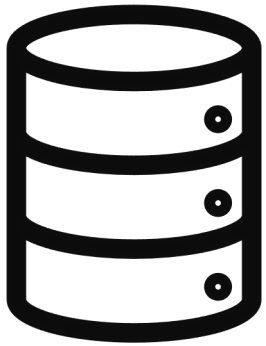
Newgame message expected format

request type

http://127.0.0.1/?req=0&who=0



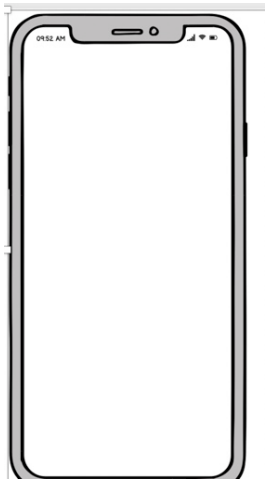


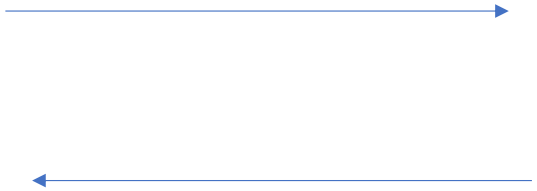


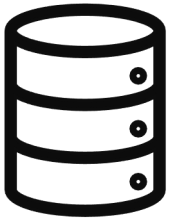

```
{
"error": false,
"msg": "NEW GAME CREATED",
"state": 0
}
```

POLLING message expected format

http://127.0.0.1/?req=1&who=0



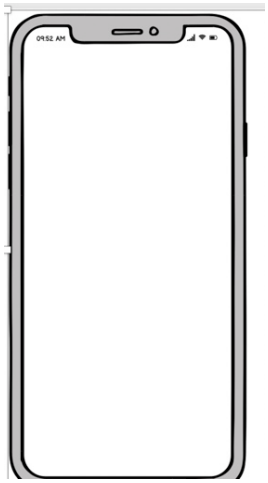


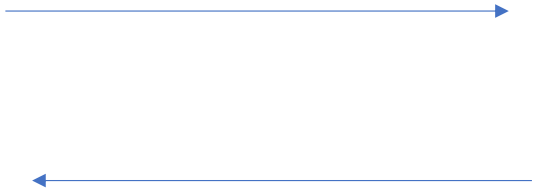


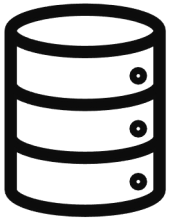

```
{
"error": false,
"move": -1,
"msg": "POLLING MESSAGE",
"state": 0,
"who": 1
}
```

NEWMOVE example

POST: http://127.0.0.1/?req=2&who=0&move=4





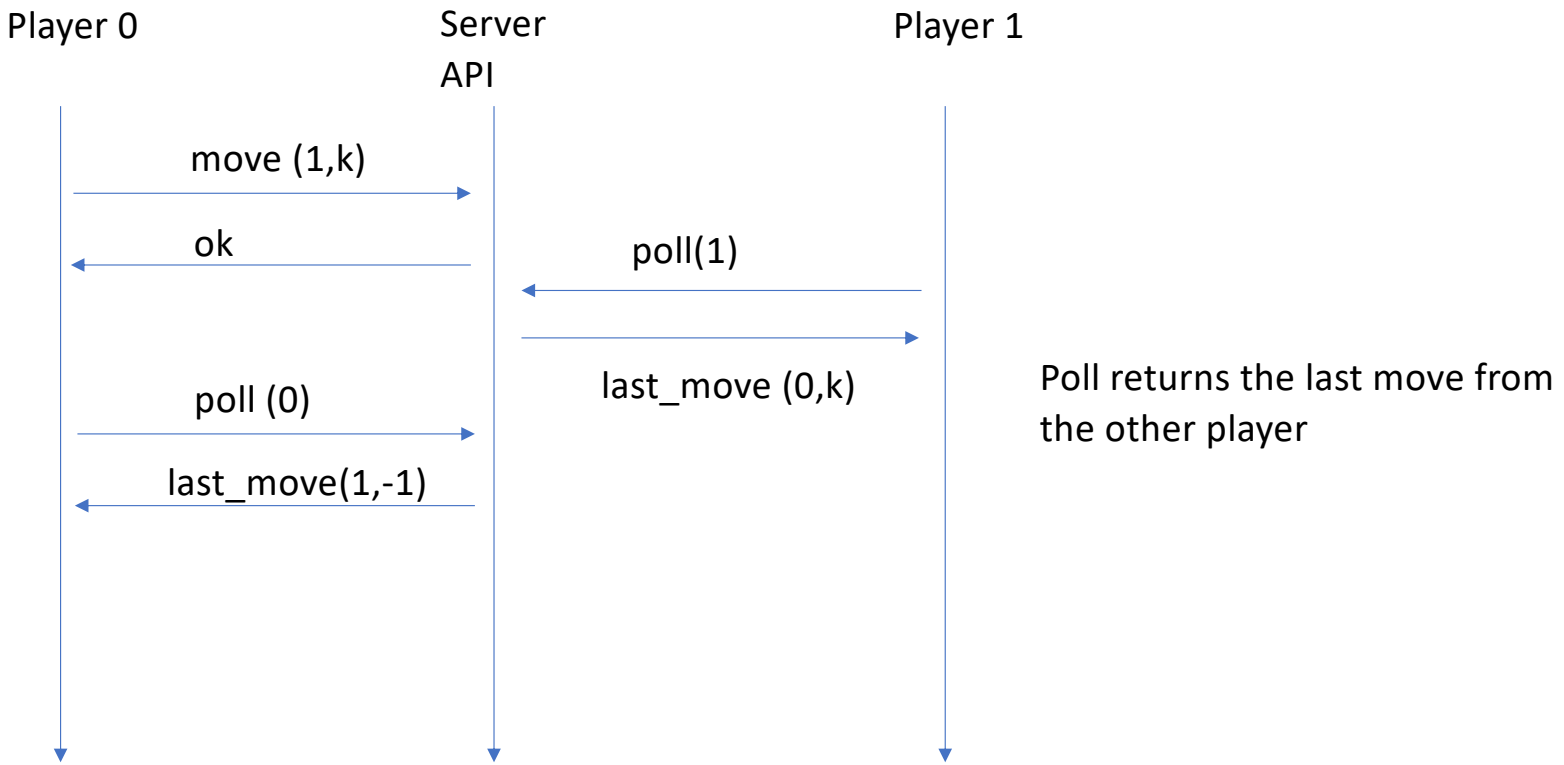



```
{
"error": false,
"msg": "NEW MOVE",
"state": 0
}
```

Writing test first (TTD) with postman

```
pm.test("Status code is 200", function () {
pm.response.to.have.status(200);
});
pm.test(”Error code", function () {
var jsonData = pm.response.json();
pm.expect(jsonData.error).to.eql(false);
});
```

Sequence diagram simple scenario

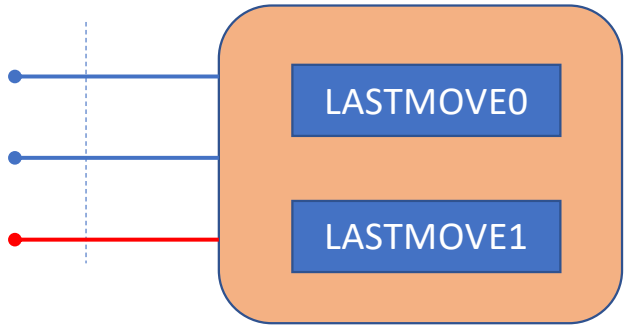


Server side model

READ LAST MOVE OF THE ADVERSARY = GET

WRITE THE NEW MOVE = POST

NEW GAME = GET




```
parserget = reqparse.RequestParser()
parserget.add_argument('req',type=int,required=True)
parserget.add_argument('who',type=int,required=True)
```

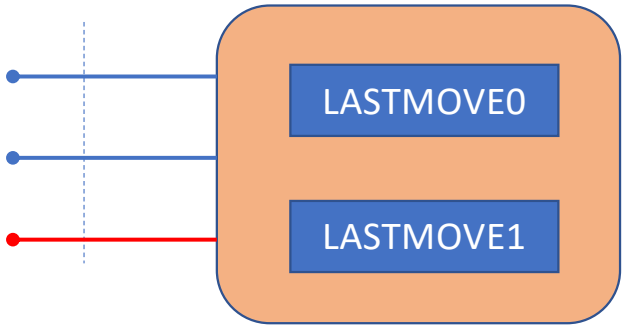
```
  def get(self):
        args = parserget.parse_args()

        req = args['req’]
        who = args['who’]

        if req==NEWGAME:
              LAST_MOVE0=-1;LAST_MOVE1=-1
              return  {'error':False,'msg':"NEW GAME CREATED"}
        if req==POLLING and who==0:
              return {'error':False,'msg':"POLLING MESSAGE","move":LASTMOVE1}
        if req==POLLING and who==1:
              return {'error':False,'msg':"POLLING MESSAGE","move":LASTMOVE0}
        return {'error':True}
```

Server side model

READ LAST MOVE OF THE ADVERSARY = GET

WRITE THE NEW MOVE = POST

NEW GAME = GET

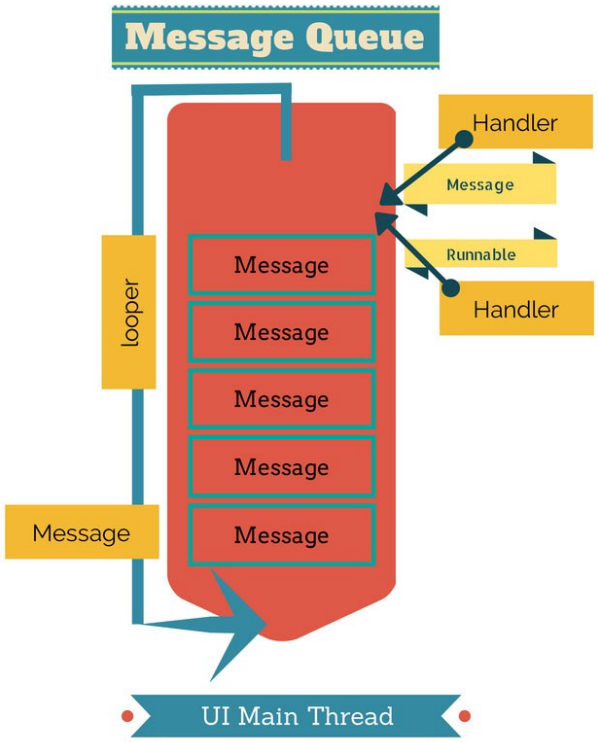



```
def post(self):
    global LASTMOVE0,LASTMOVE1
    args = parserpost.parse_args()

    req = args['req']
    who = args['who']
    move = args['move']
    if req==MOVE and who==0:
        LASTMOVE0=move
        return {'error':False,'msg':"MOVE MESSAGE"}
    if req==MOVE and who==1:
        LASTMOVE1=move
        return {'error':False,'msg':"MOVE MESSAGE"}
    return {'error':True}
```

Client side model

- Periodic Polling
- Update the screen according to the result

Android handlers

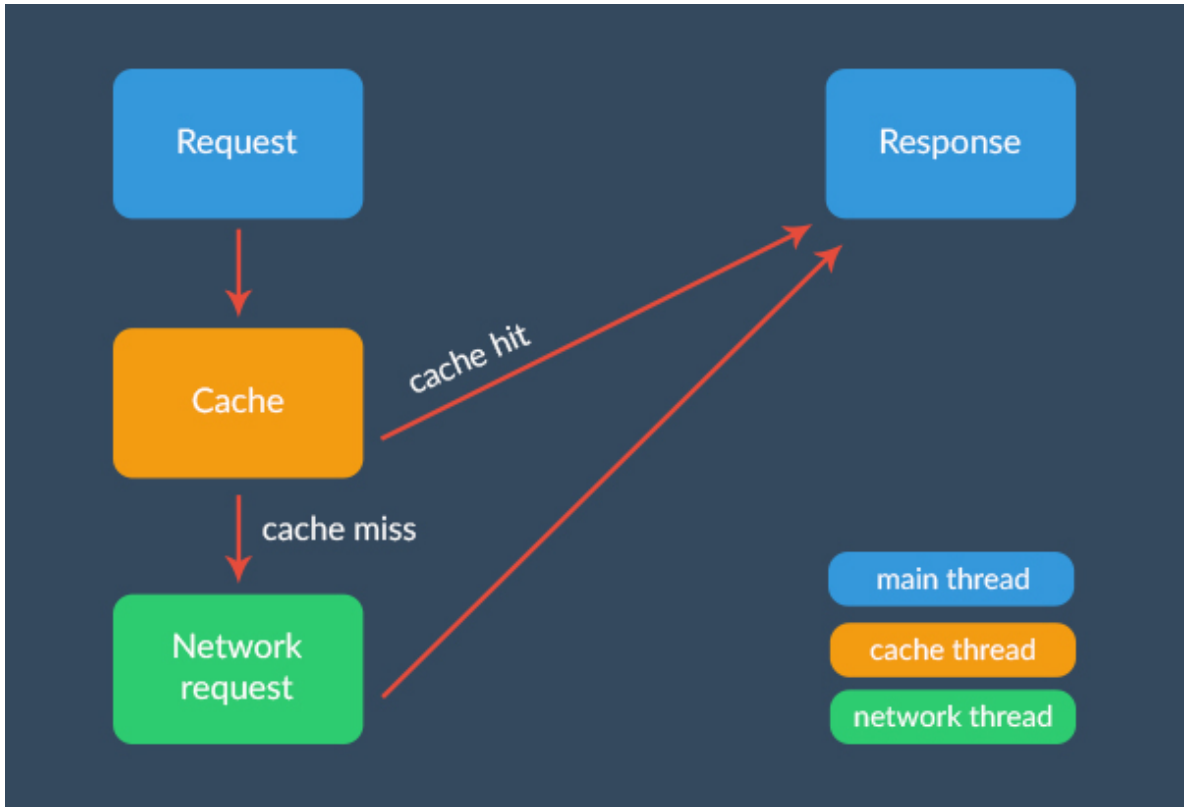



```
Handler(Looper.getMainLooper()).postDelayed({checkMove()},pollingperiod)
```

```
fun checkMove() {
//Poll the server
//Invalidate()
//Schedule another Handler message
}
```

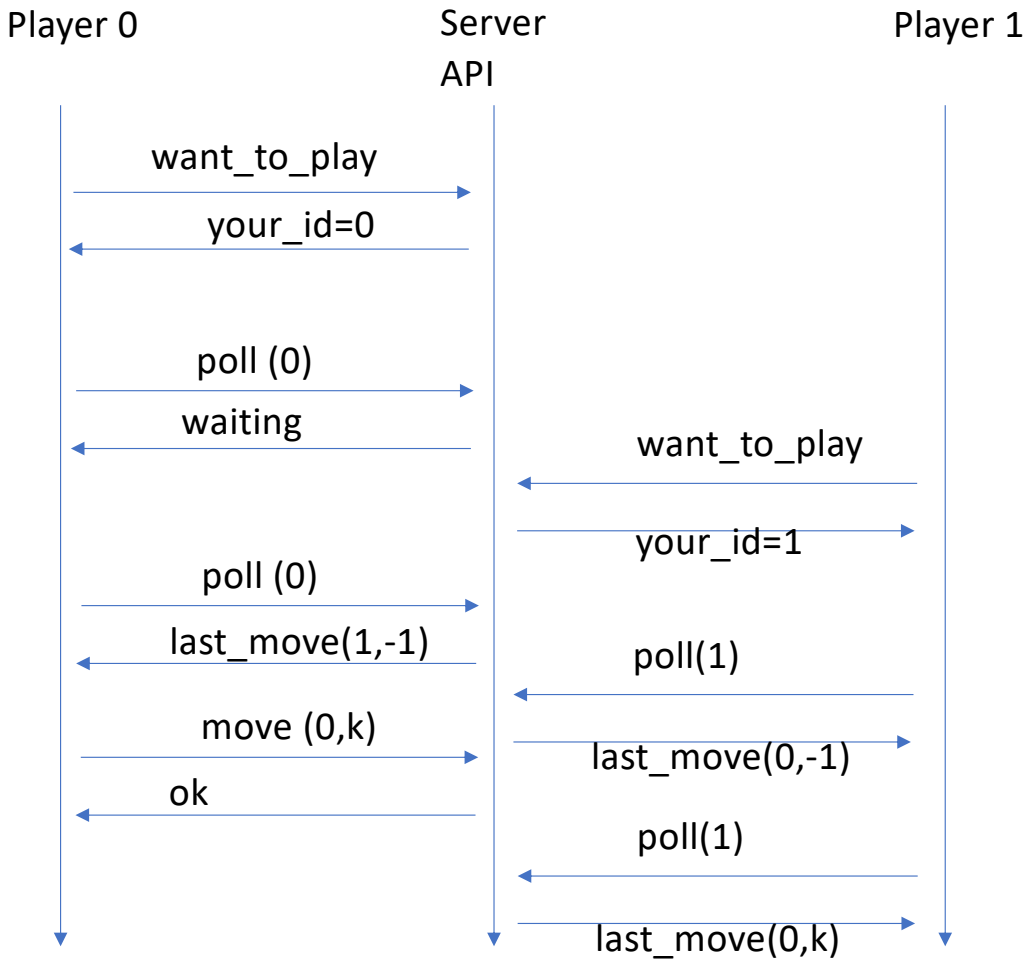
Credits: https://en.proft.me/2017/04/15/understanding-handler-android/

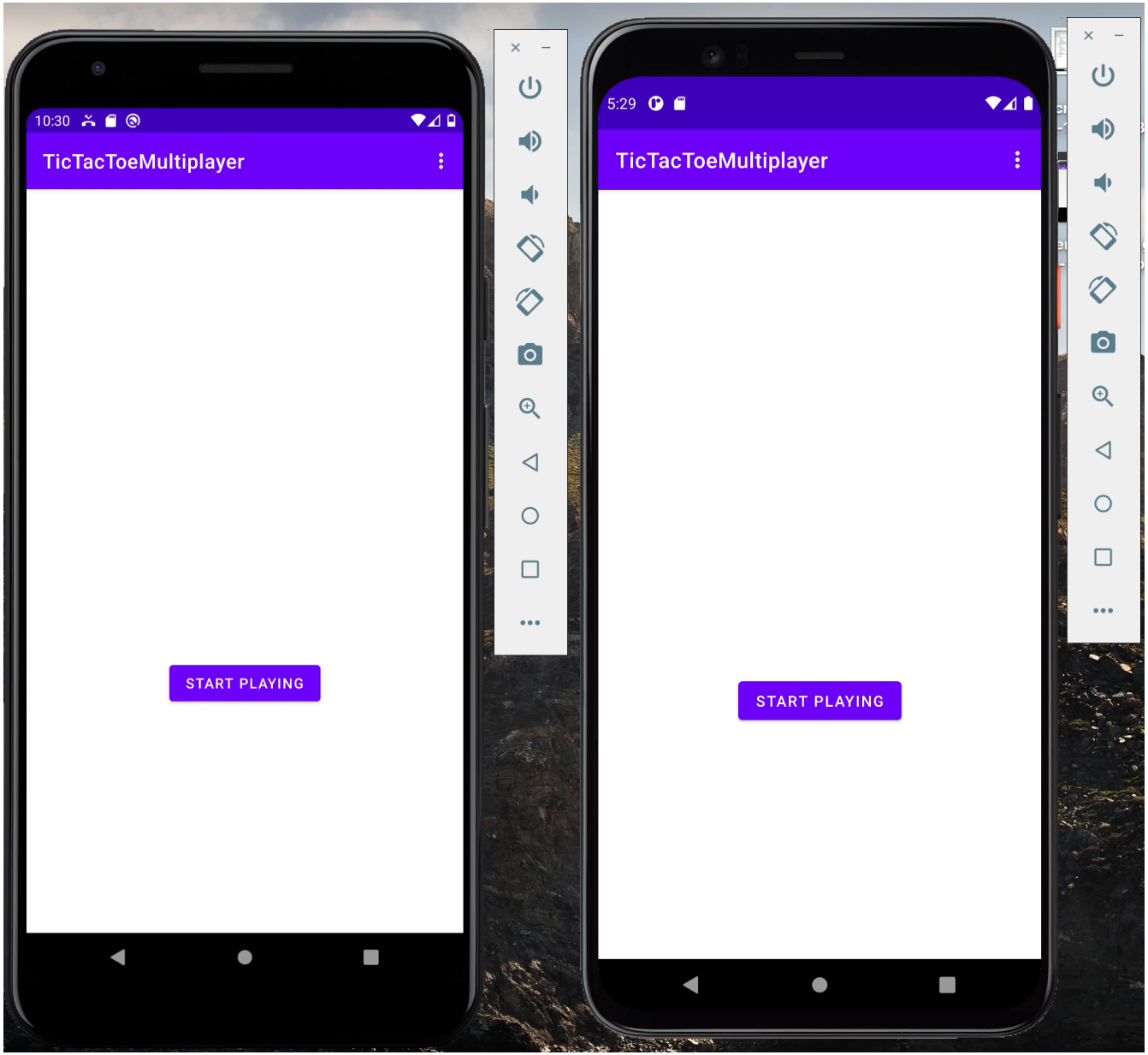
Making HTTP calls with volley



See: https://developer.android.com/guide/topics/connectivity

Sequence diagram two players

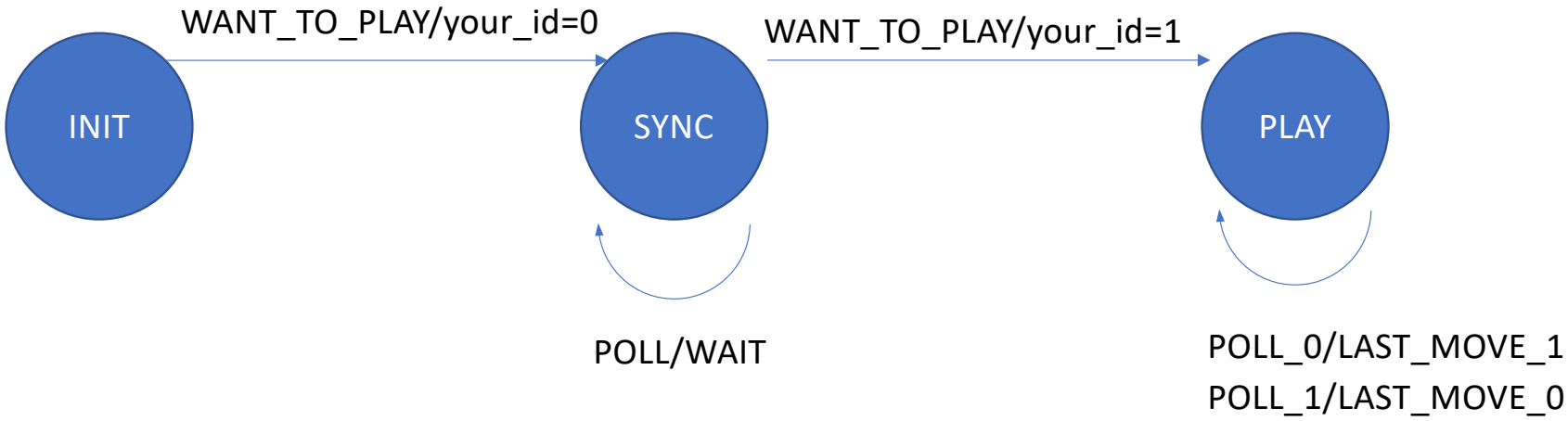



10:30
TicTacToeMultiplayer
START PLAYING
5:29
TicTacToeMultiplayer
START PLAYING

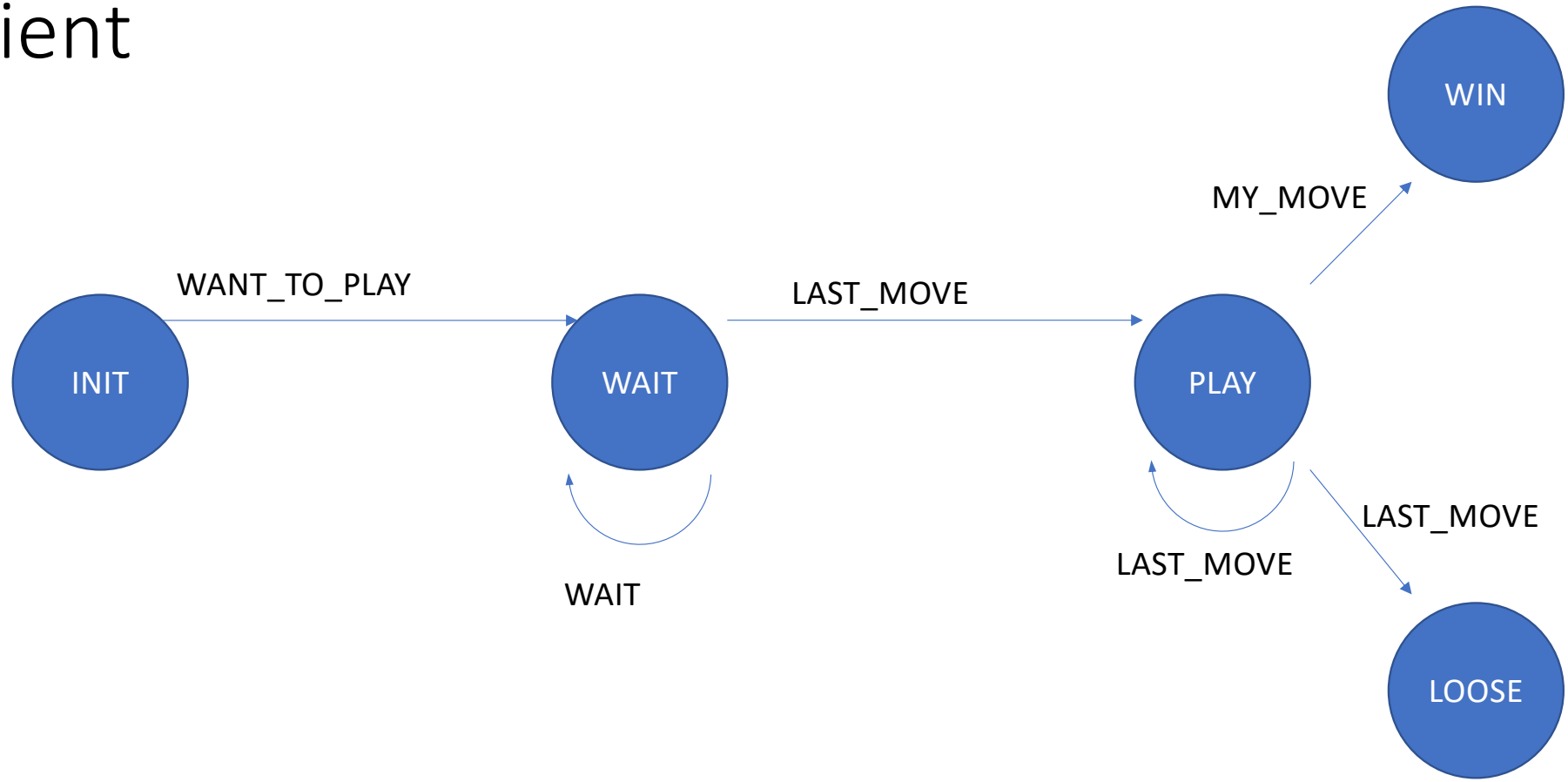
Challenges

1. Managing synchronization among players
2. Managing authentication
3. Managing authorization to perform a move

Server


WANT_TO_PLAY/your_id=0
WANT_TO_PLAY/your_id=1
INIT
SYNC
PLAY
POLL/WAIT
POLL_0/LAST_MOVE_1
POLL_1/LAST_MOVE_0

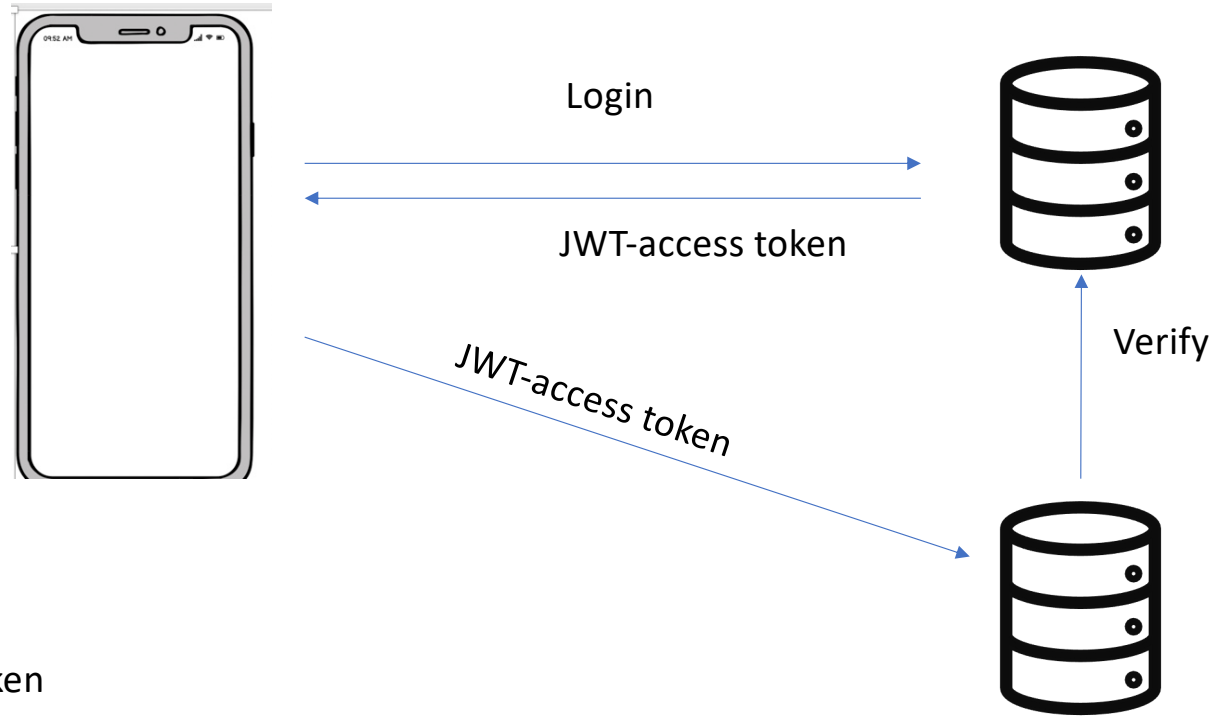
Client


INIT
WANT_TO_PLAY
WAIT
WAIT
LAST_MOVE
PLAY
MY_MOVE
WIN
LAST_MOVE
LAST_MOVE
LOOSE

Securing the app

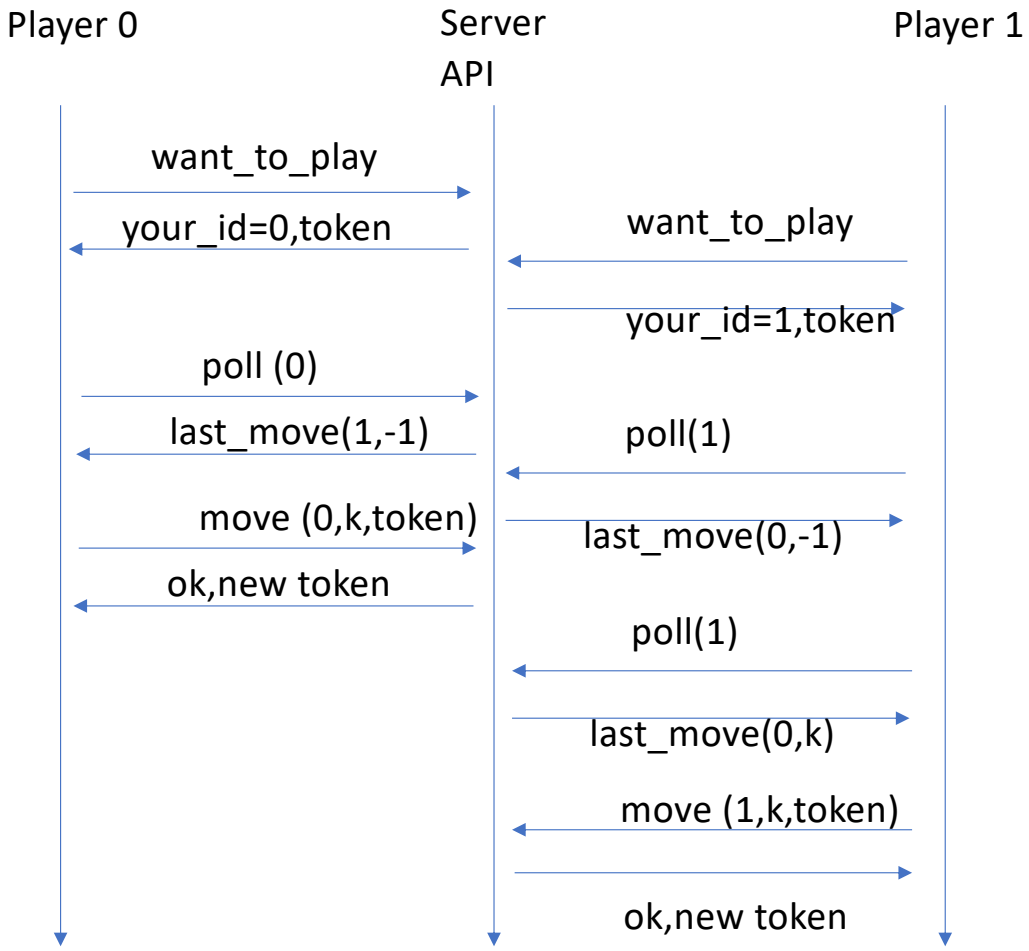
- Use a JWT access-token to enter the app
- Use one-time JWT to authorize the next move

Access



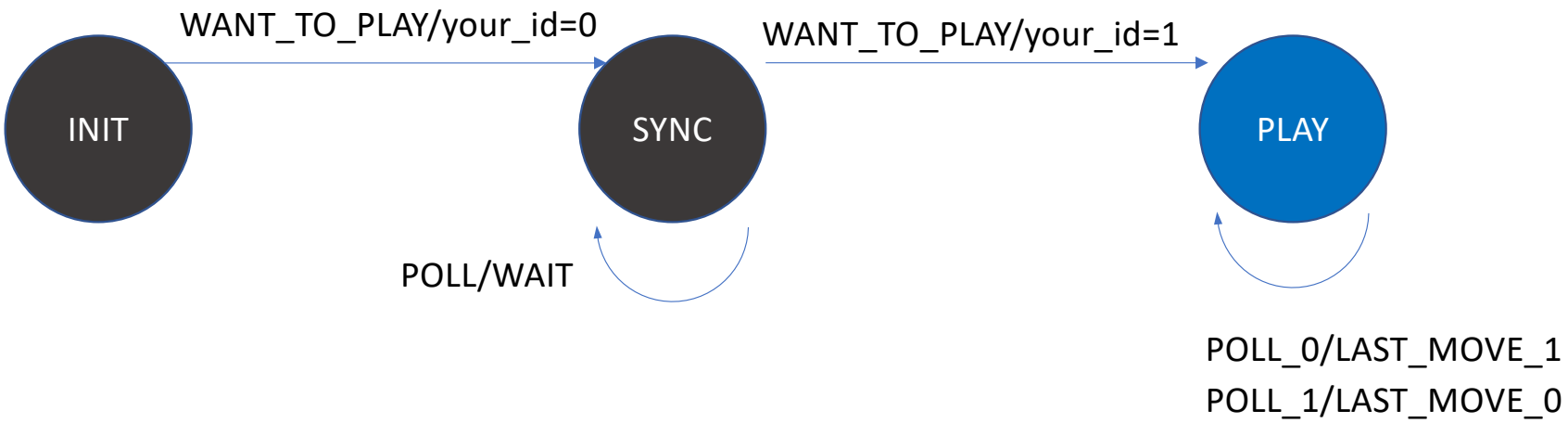
Refresh token

Managing authorization to make the next move



Flask decorators

Server


WANT_TO_PLAY/your_id=0
WANT_TO_PLAY/your_id=1
INIT
SYNC
PLAY
POLL/WAIT
POLL_0/LAST_MOVE_1
POLL_1/LAST_MOVE_0