LLD - Design a parking lot

Management System

- Parking Lot
- Book My Show
- Splitwise

- Email Campaign*

Design a parking lot

↳ Design

- Requirement → API

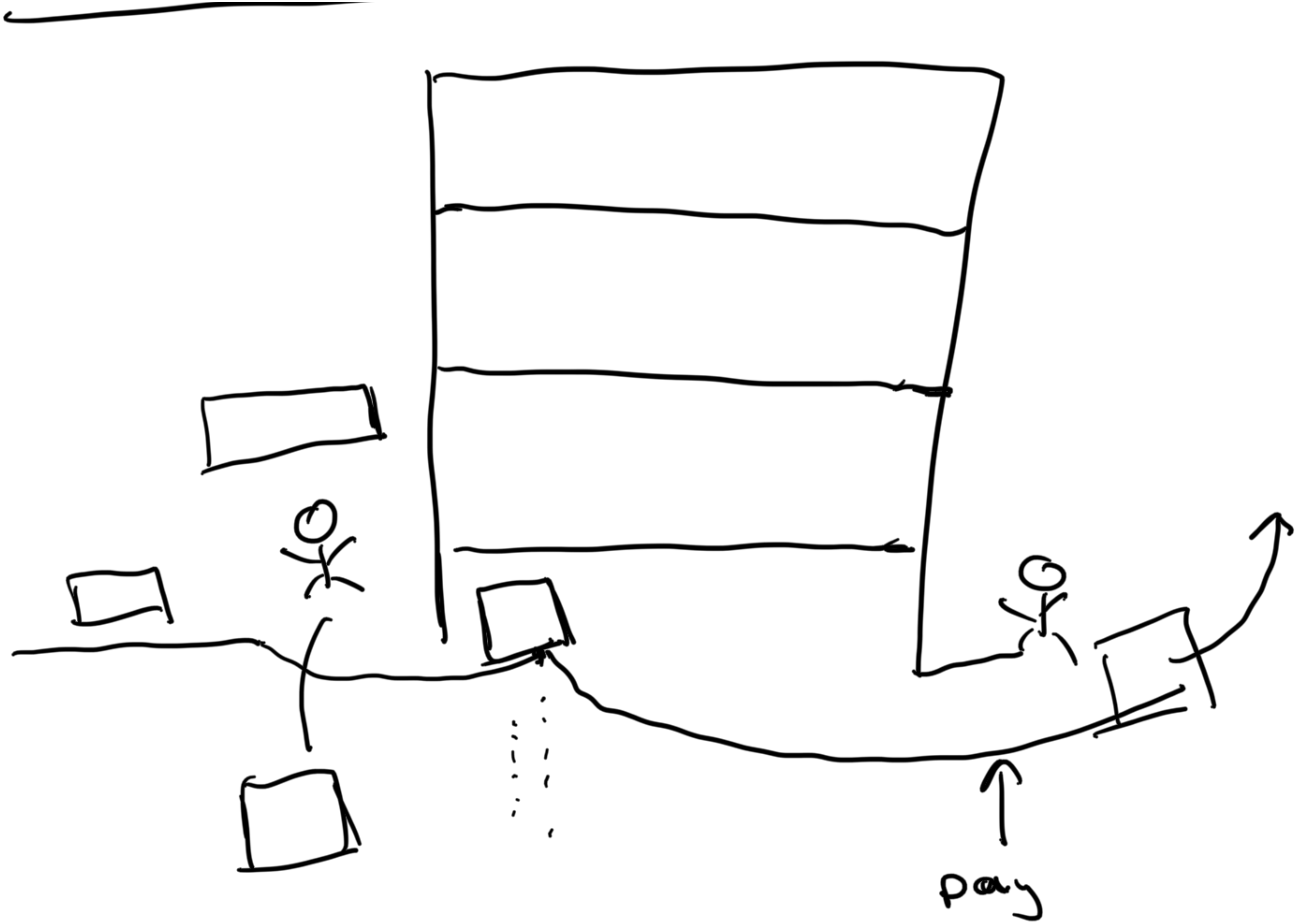
- Use case design
- Class Diagram
- Sequence Diagram

→ Implementation

- → CLI - command
- → API
- → Controllers
- → Services
- → Repository

} In memory

Parking lot



Requirement Gathering

Questions → Requirements

Curent scope
Futue scope } What ....? Can .....?

Behaviour } How ....?

---

1. Each slot can be in various states
   → AVBL
   → OCCUPIED
   → Out of service

1. Can a PL have multiple floors?

   Yes — N floors

2. Each floor will have multiple slots

3. Slots
   - → Large
   - → Medium
   - → Small

   } —

   — Four

   — 2 wheeler

   — 3 wheeler

4. A vehicle can only park

in its own type slot

5. Multiple entry gates & exit gates

6. Each entry gate will hae a display board

7. How do we enter?

→ Operator will give you a ticket

8. How do we pay?

Calculation

→ cost(type, duration)

$50 + 80(t-1)$

→ cost(type, duration, entrytime)

Modes

- Cash

- Credit / Debit }
- Online

Payment counter

- At every floor.

Display Board

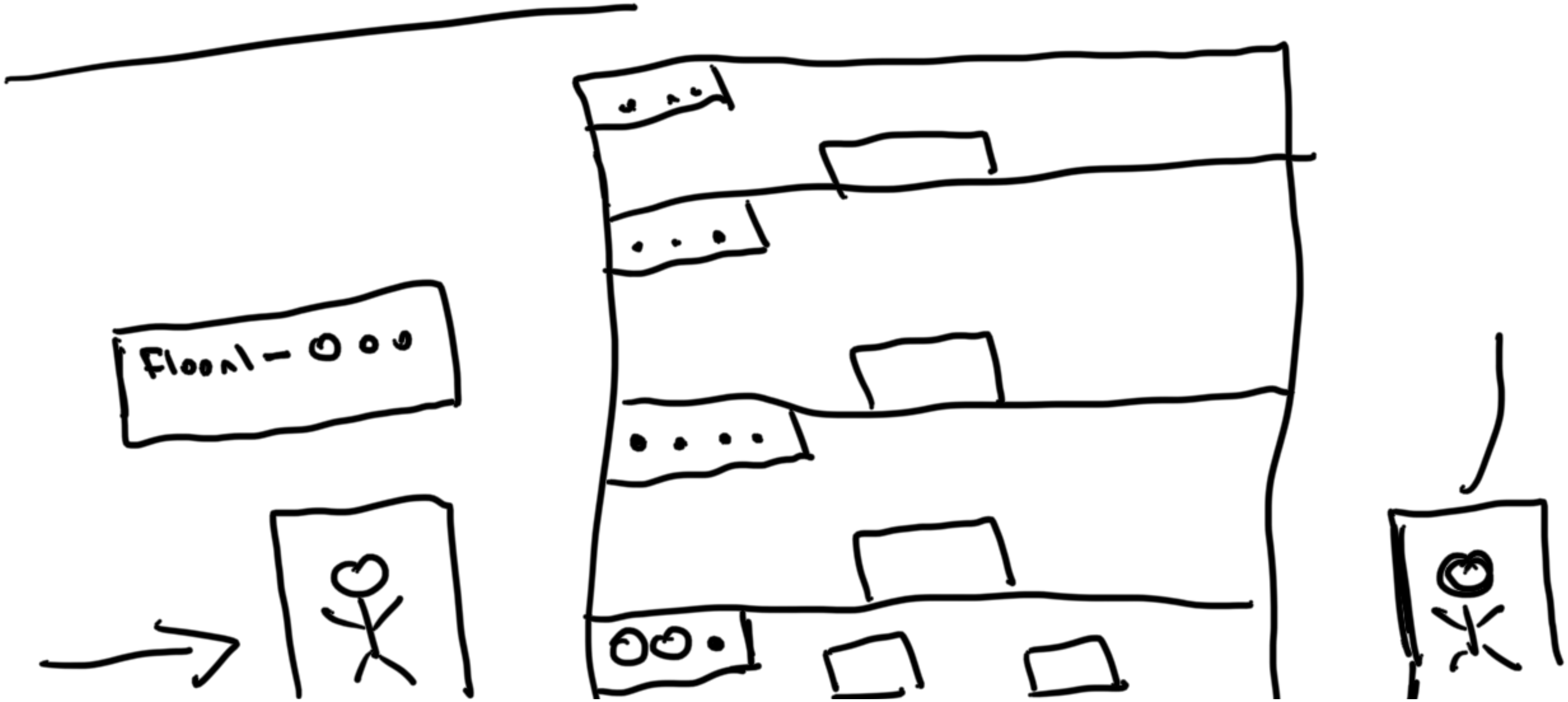
- Empty slots

Allocate

→ any random empty slot

→ Future might want to make this more optimal

Visualisation



1. Give me a slot
2. Allocate a slot
3. Give you a ticket
   - Vehicle No
   - Entry time

     ....

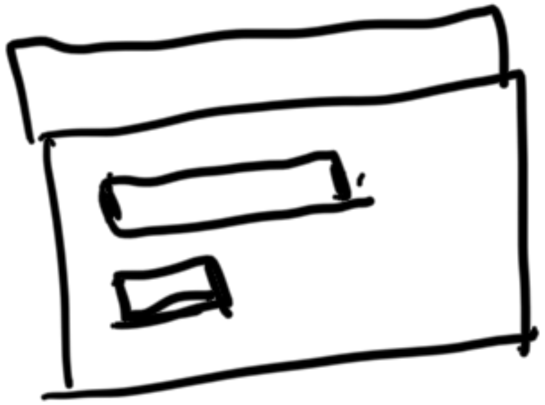
Actors

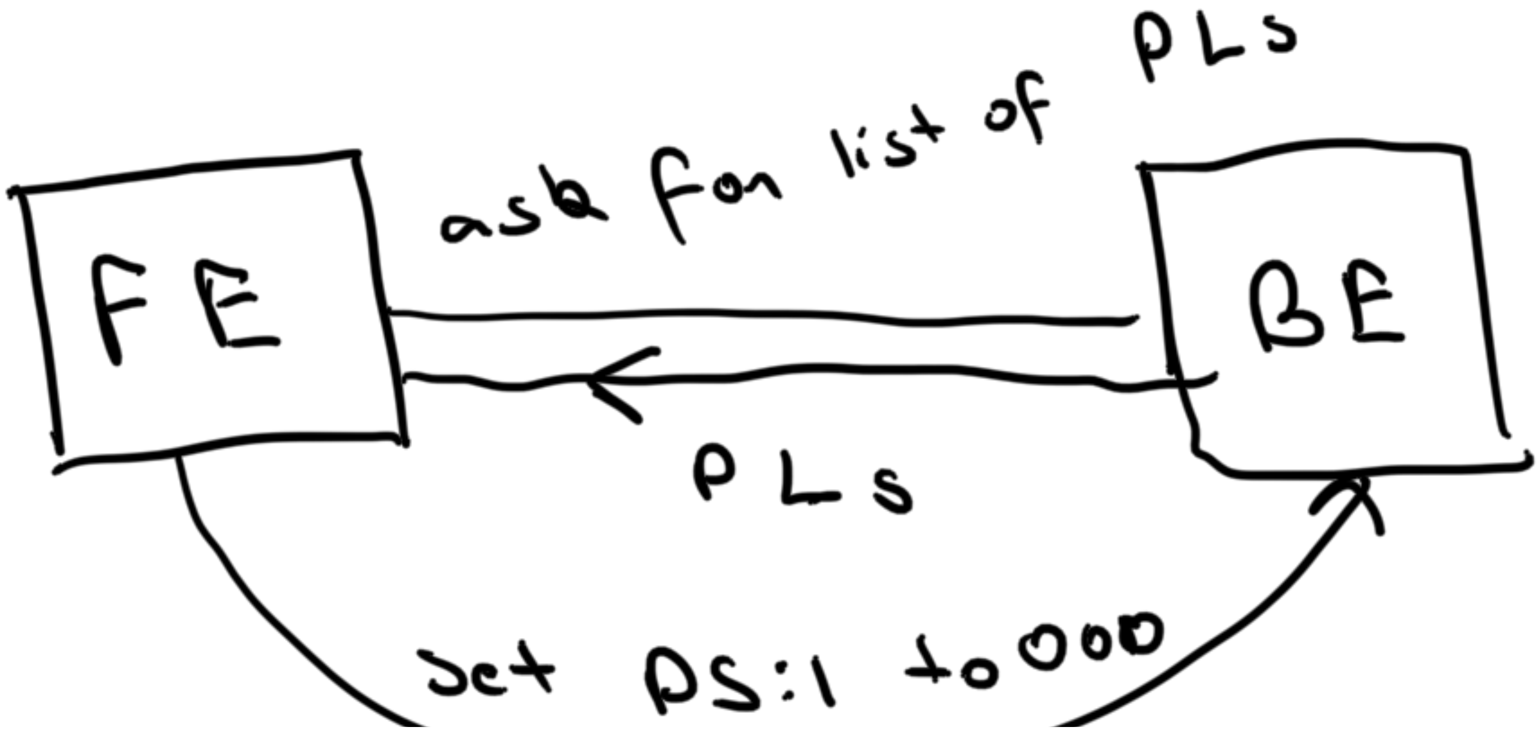
6:07 – 6:10

– 10:40

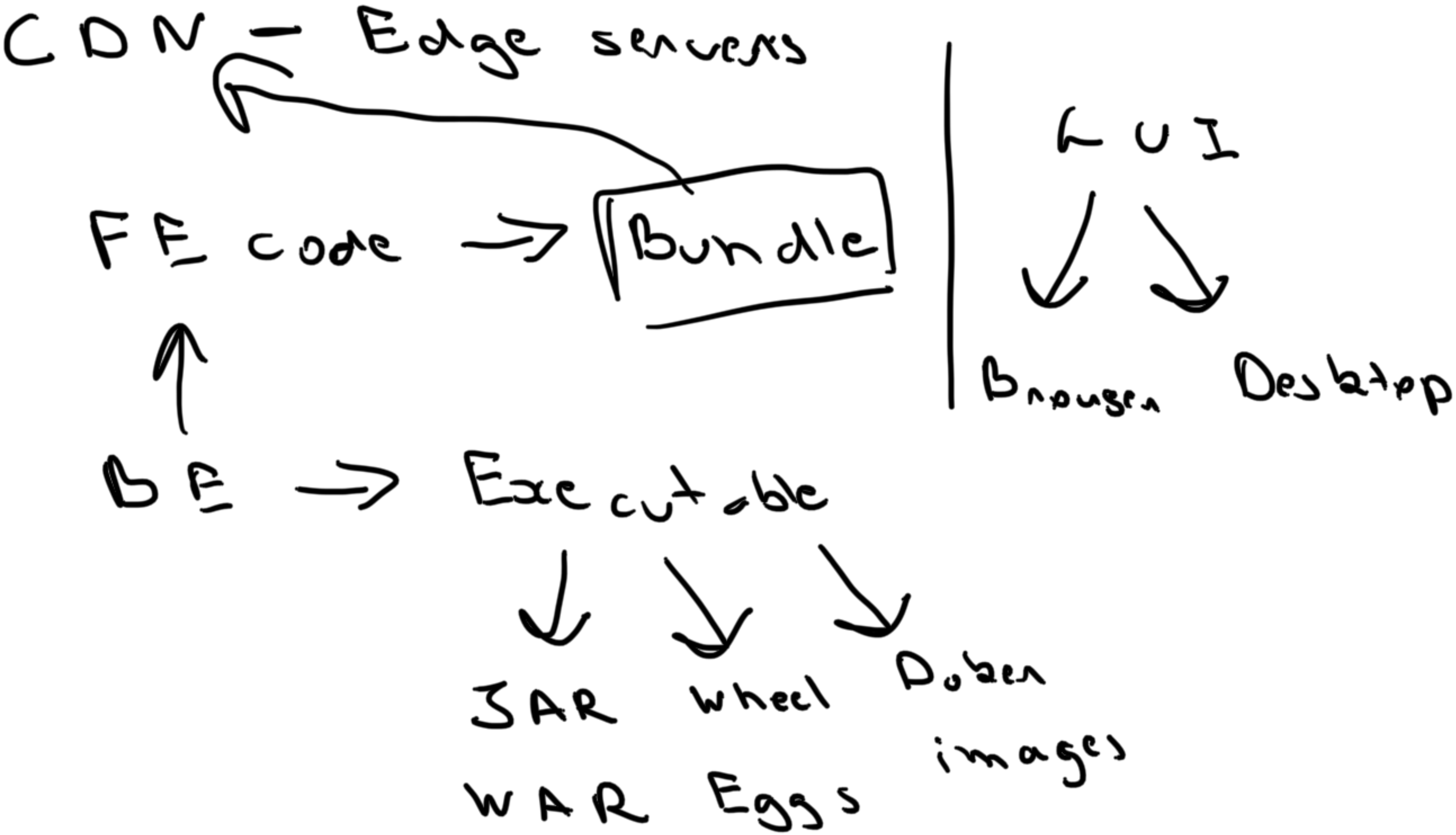
API design

Admin





Done



CDN — Edge servers

FE code → Bundle

BE → Executable

JAR WAR

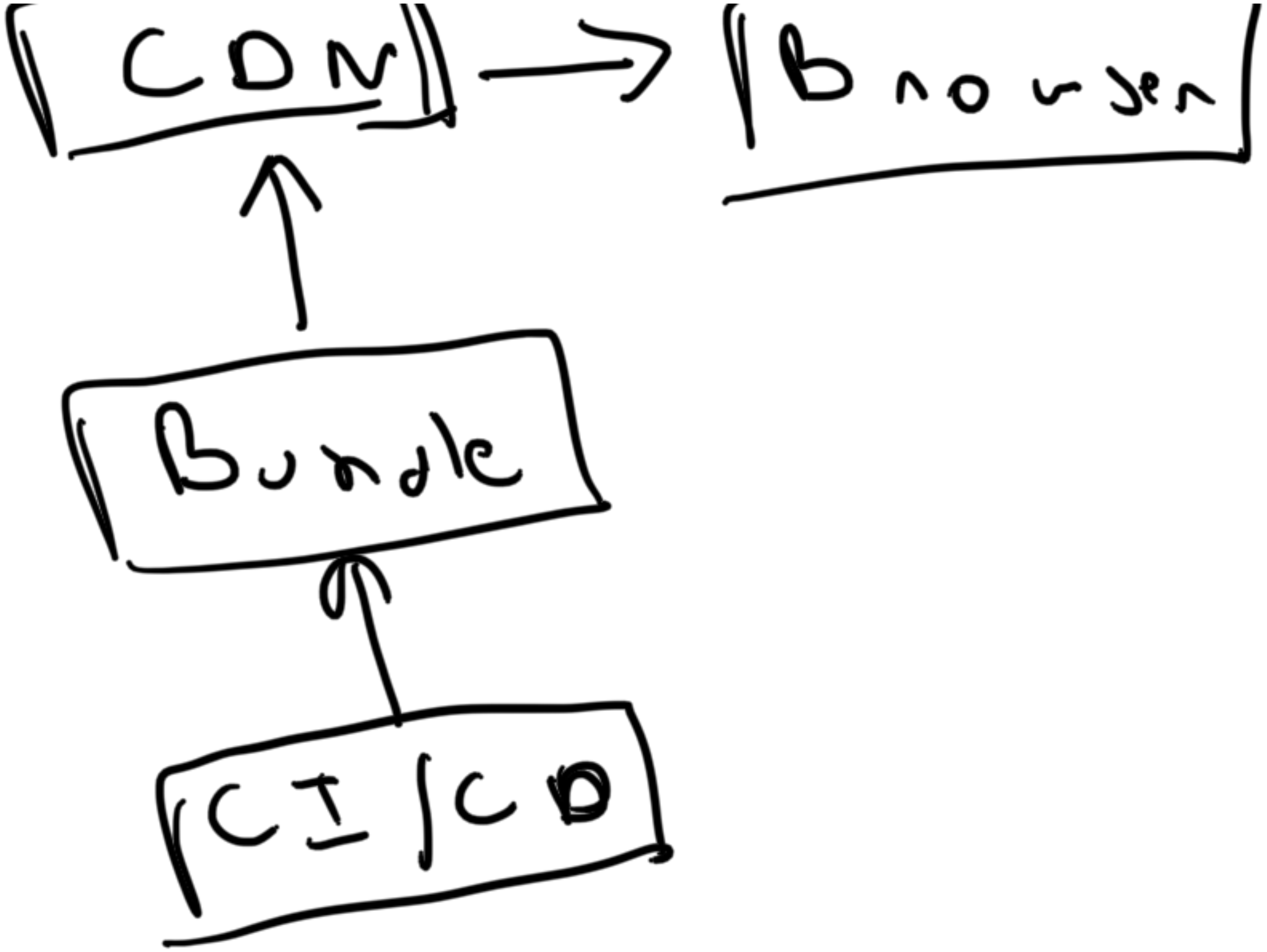
wheel Eggs

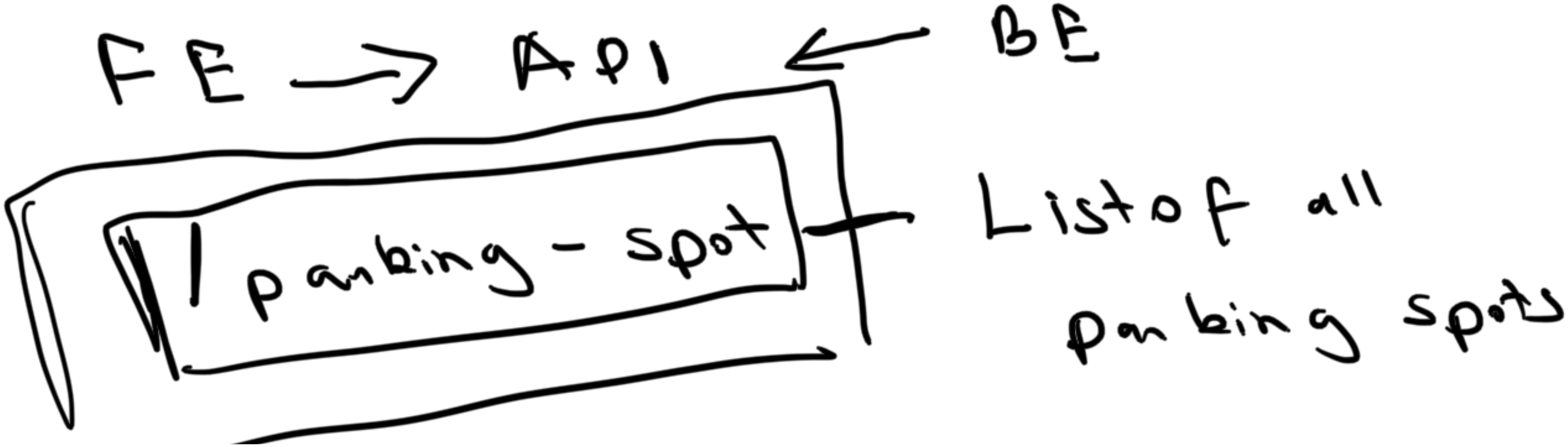
Docker images

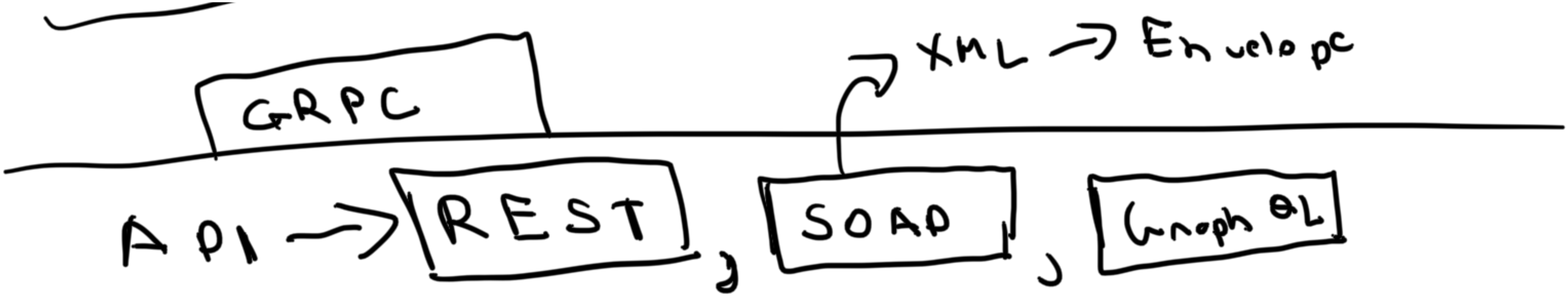
GUI

Browser

Desktop


CDN
Browser
Bundle
CI/CD


FE
API
BE
/parking-spot
List of all parking spots



C

R

U

D

Parking Lot → Resource

Parking Spot → Resource

id-1

VERB /resource/{resourceId}

/parking-spot/1

CRUD

C — Post | Put

R — Get Post

U — PUT | PATCH

D — DELETE

POST /parking-spot ②

```
[ {
      name: Parking Lot 1

  } ]
```

ID Not idempotent

input → output

```
PUT /parking-spot/:id
    {

    }
```

Put vs Patch

└→ Complete update

→ Update parking status

PUT /parking-spot/:id/status

PUT /ps/:id

```
{
    name: 'New Name'
}

{
    name:
    status: NULL
```

PATCH /ps :id → Path parameter.

```
[ {
      field : name
      op : set |unset | concat
  } value : "New Name"
]
```

Post – Creating

– Operation

GET /parking-spot ?staus= AVBL

query

{

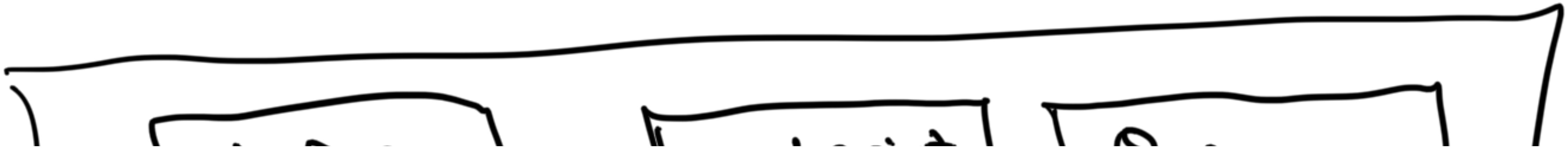
}

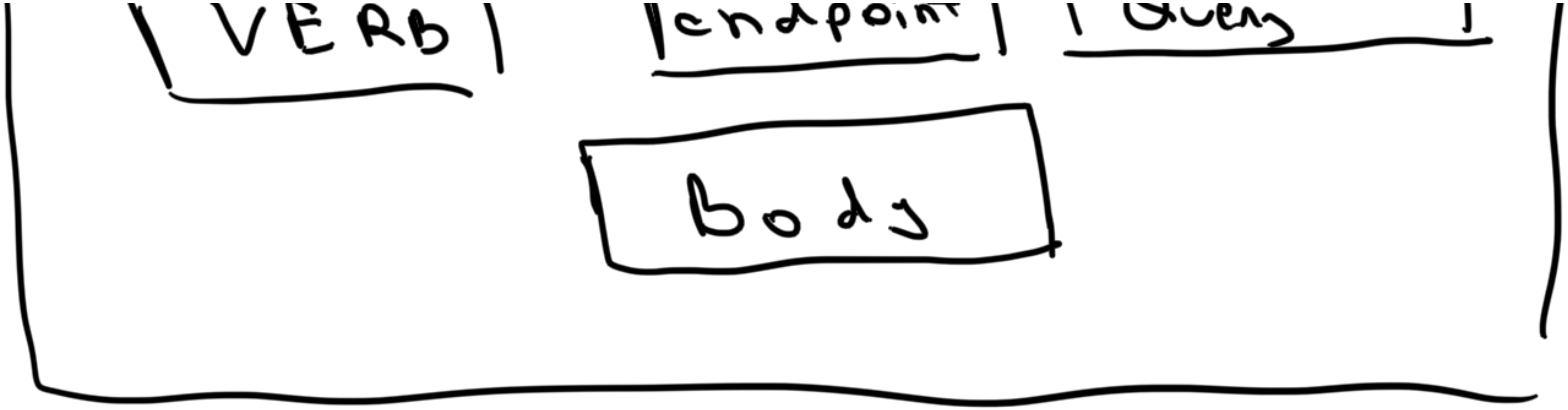
| search — Post

---

① Api Design

Use ease → AP Design





→ Swagger → code

→ Postman

Monday
- Parking
- Admin

② Class Diagram

Tuesday

③ Models

- Entity Classes
- DTOs

• • •

Controller, Service, Repo.

Command

REST

↳ Representational

State

Transfer

→ /login

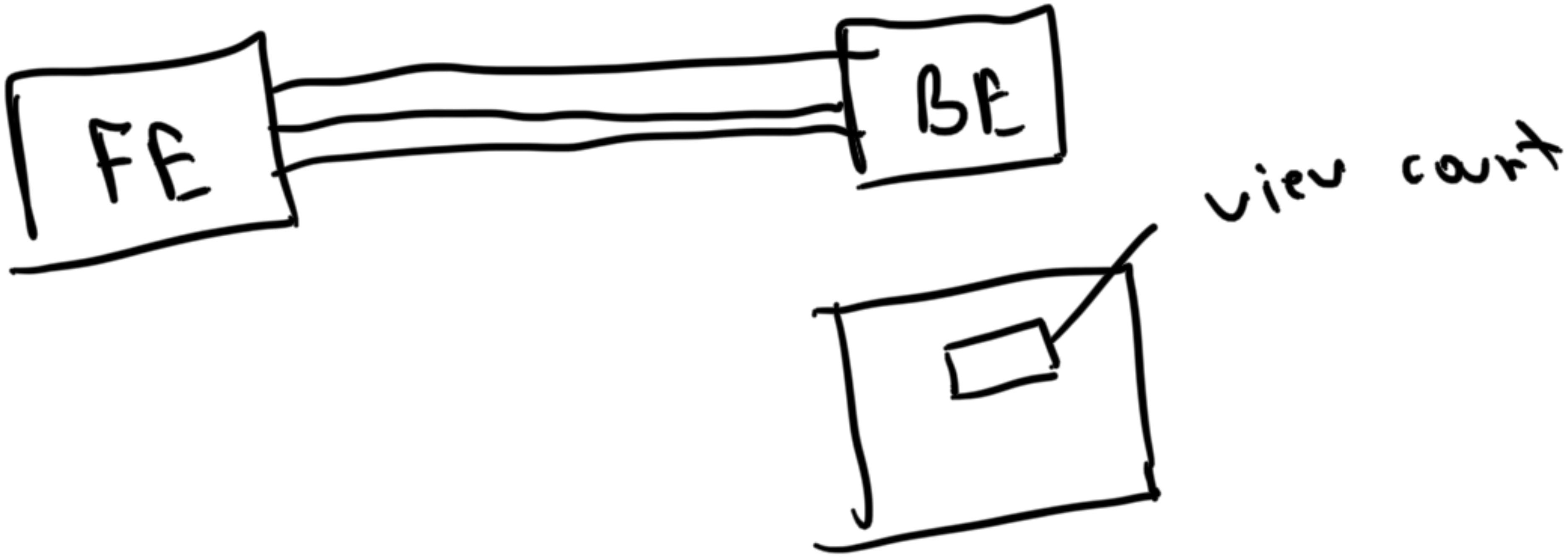
→ /parking-spot/1/ ___

{

}

| session

ip address



JWT → User

— password

- email

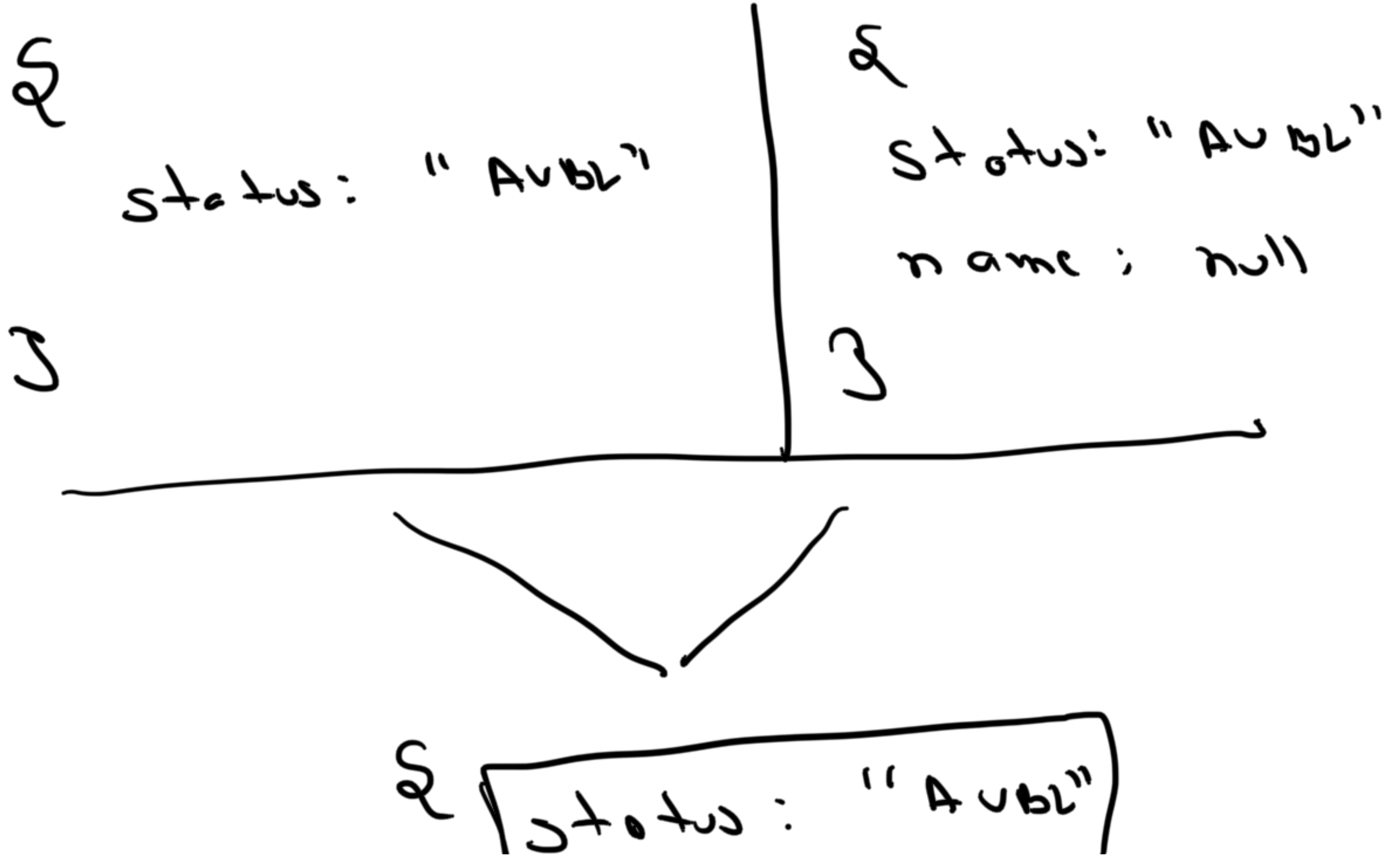
<u>DB</u>

```
{
  name: "Name"
  status: "AVBL"
}
```

PUT

```
{name: null
 status: "OOO"
}  ——>  DB   name = null

if (value !== null)
```

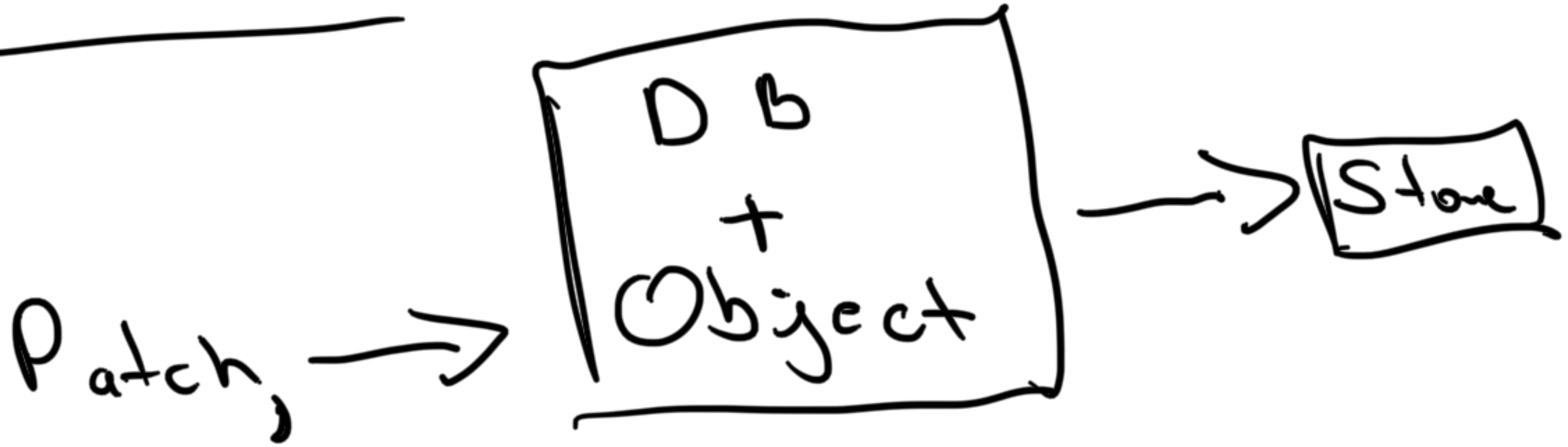


```
{                          |  {
    status: "AVBL"         |    status: "AVBL"
                           |    name : null
}                          |  }
```

```
{ status : "AVBL"
}      name : null
```

```
{
    name: null
}
```

--no-verify



Existing

Object