Brainstorming

3 Primary Parts

1: Tenant Logic - Random request, get on elevator, get off elevator } "Backend"
2: Elevator Logic - Pick up riders, drop off riders
3: User Visuals - Display floor, display movement, display doors open/closed

Delivery Time: Tommorrow (Thursday) night

Plan Overview

1: Tenant & Elevator Logic/Rules outline
2: Class pseudocode for Tenant & Elevator
3: UI wireframing
4: UI Component pseudocode/outline
5: Typing outline
6: Setup Programming environment & Repo
7: Scaffolding
8: Tenant & elevator code
9: Front End/UI CSS
10: Testing
11: Documentation & polish

Tenant Rules

1: A tenant is spawned in a starting floor with a random id that is ≥ 1 & ≤ 20

2: A tenant is spawned with a destination floor id that is both:

a: $\neq$ to its starting floor

b: ≥ 1 & ≤ 20

3: A tenant is spawned with Direction determined by:

0 (down): Destination floor $<$ Starting floor

1 (up): Destination floor $>$ Starting floor

4: A tenant is spawned with a request datetime indicating when they were spawned

5: After the tenant is initialized, it is pushed onto the pending list

Elevator Rules

1: An elevator has 3 Directions (0:Down, 1:Up, 2:Neutral)

2: An elevator has a passenger queue it sorts based on direction

3: An elevator has a next Destination floor id that shows its current intention

4: When an elevator passes a floor, it checks if it is its next destination floor & if there are any passengers with the same direction, & if neither is true it passes the floor.

5: When an elevator stops at a floor, it does the following

1: opens its doors

2: Removes any passengers from queue w/ a matching destination floor

3: Calculates new direction (may be the same)

4: Picks up any passengers w/ matching direction & sorts them

5: Calculates new Destination floor

6: Closes doors

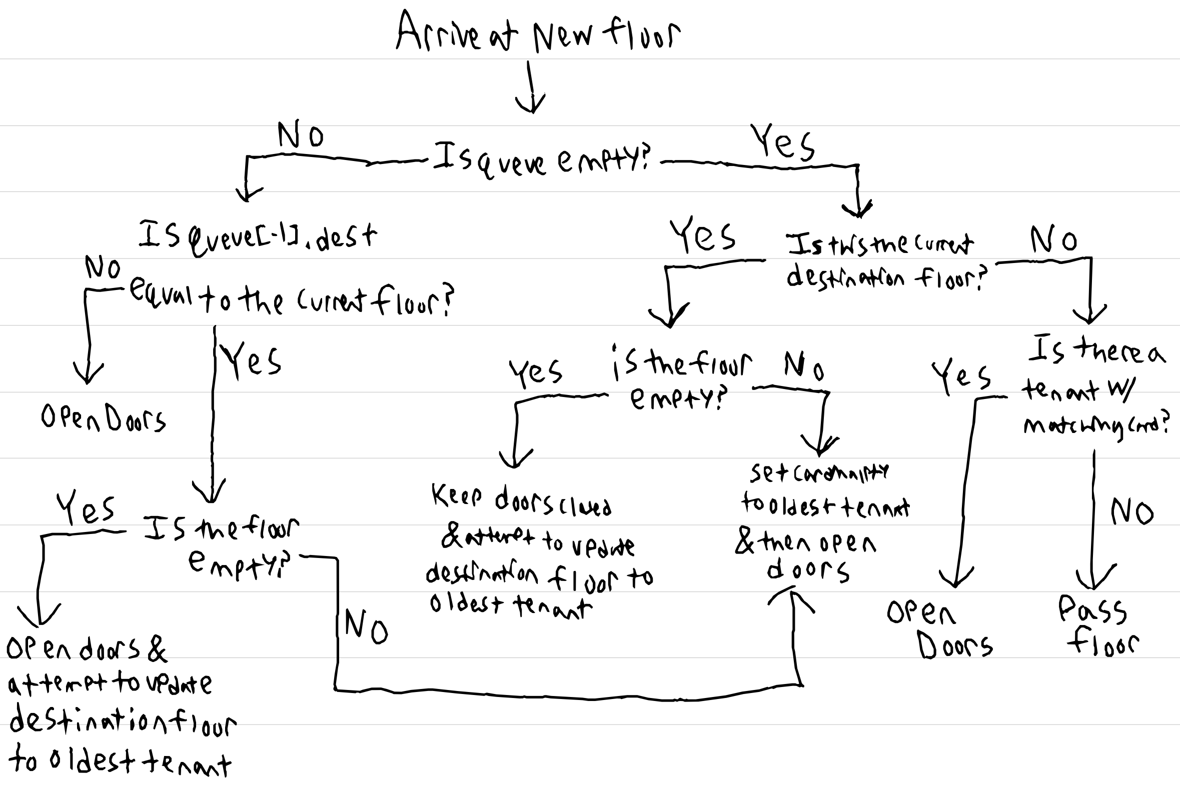
6: An elevator calculates its new direction as the first of the following

1: Checking the direction of the first passenger in queue

2: Checking the direction of the oldest tenant on the floor

3: The direction to approach the oldest tenant

7: An elevator calculates destination floor as either the destination of queue[0] or the starting floor of the oldest tenant



Arrive at New floor
Is queue empty?
No
Yes
Is queue[-1].dest equal to the current floor?
No
Open Doors
Yes
Is the floor empty?
Yes
Open doors & attempt to update destination floor to oldest tenant
No
Is this the current destination floor?
Yes
is the floor empty?
Yes
Keep doors closed & attempt to update destination floor to oldest tenant
No
set cardinality to oldest tenant & then open doors
No
Is there a tenant w/ matching card?
Yes
Open Doors
NO
Pass floor

Maybe open has 3 Stages

Exit, enter, & Sort

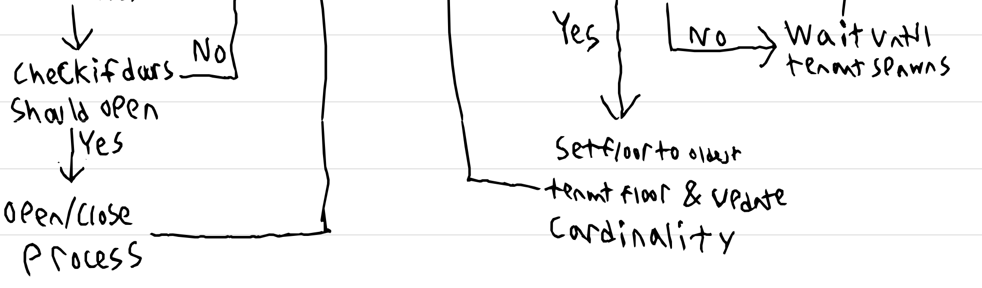
Primary Processes

1: Check if doors should open on floor

2: Determine Next floor

3: Determine Cardinality

General Flow (starting at doors closed)



Elevator methods

addRequest: Creates & adds new tenant to queue

stopAtCurrentFloor: Stops at current floor & performs various tasks

dropOffPassengers: Drops off current passengers matching floor

PickUpPassengers: Picks up passengers matching direction

sortPassengers: Sorts passengers by floor according to direction

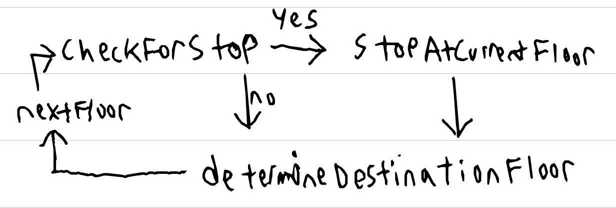
determineDirection: Determines new value for direction

determineDestinationFloor: Determines new destinationFloor

nextFloor: Increment/Decrement floor according to direction

CheckForStop: Determine if the current floor should be stopped at

General Flow



Cronjob every 10 seconds

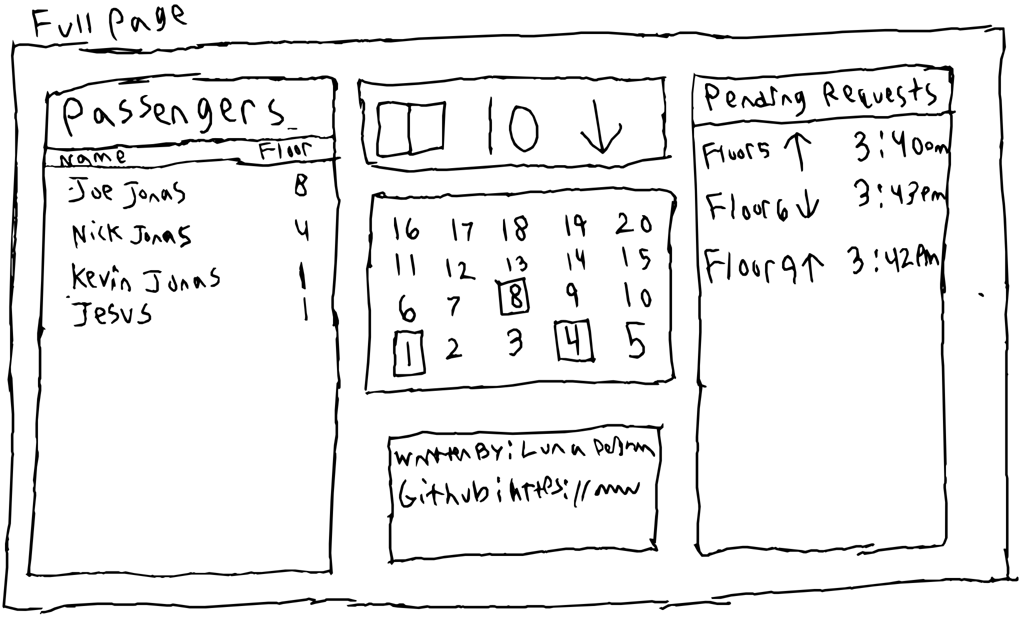
addRequest

Tenant & Elevator Class Pseudo code

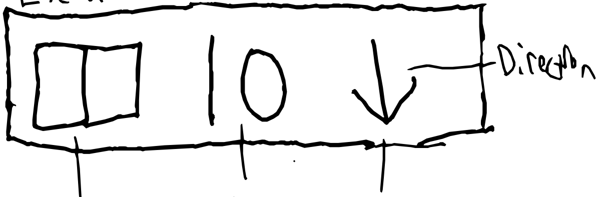
```
Class Tenant():
   name: String
   StartingFloor: number
   destinationFloor: number
   direction: Direction Enum (0,1,2)
   requestTime: DateTime (luxon)

   Constructor():
      name = random name
      startingFloor = random floor (≥1 & ≤20)
      destinationFloor = random floor (≥1 & ≤20 & ≠ startingFloor)
      direction = startingFloor > destinationFloor ? 1 : 0
      requestTime = DateTime.now()

Class Elevator(): → Redux?
   direction: Direction Enum (0,1,2)
   CurrentFloor: number
   destinationFloor: number
   Passenger Queue: Tenant[]
   request Queue: Tenant[] → To keep single source of truth

   addRequest():
      requestQueue.push(new Tenant())

   dropOffPassengers():
      Passenger Queue.find&remove all(destinationFloor = this.destinationFloor)

   PickUpPassengers():
      pickupPassengers = request Queue. Pop(where direction = this.direction)
      PassengerQueue = [...passengerqueue, ...PickupPassengers], SortBy Direction

   checkForStop():
      determine if elevator should stop at current floor or not

   PerformFloorStop
```

Wireframing

Full page



Elevator Display



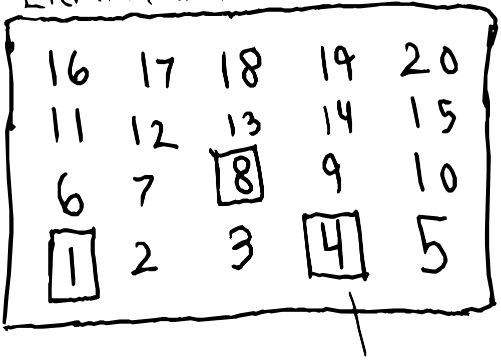
Direction

Open or close Indicator

Floor

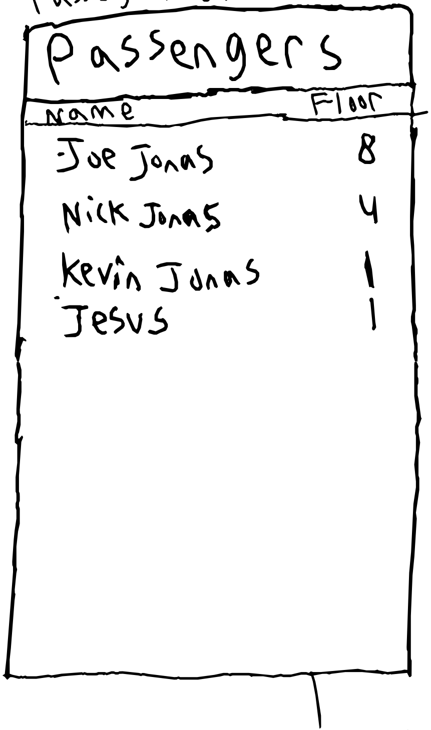
Arrow will be green & possibly animated when moving, & red/static when not

Elevator Panel



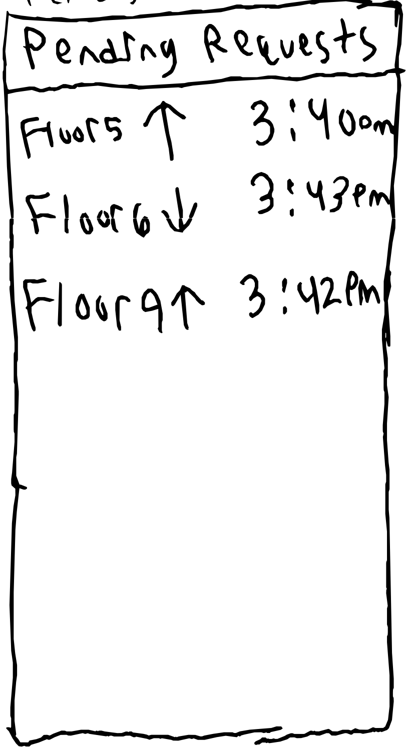
Selected floors will have a soft blue or Yellow bubbling

Passenger List



Alternating color table rows

Pending Requests



Component Breakdown

Structure

```
ElevatorControl
 ├ElevatorDisplay - Floor, door, & directiondisplay
 │    ├ElevatorDisplayArrow
 │    └ElevatorDisplayDoor
 └ElevatorPanel - Panel w/ floor buttons
      └ElevatorPanelButton
PassengerList
 └PassengerRow
PendingRequestsList
 └PendingRequestRow
AuthorCard - Static
```

Props/Typing

ElevatorDisplayArrow
- direction: Direction Enum (0,1,2)
- floorState: FloorState Enum

ElevatorDisplayDoor
- floorState: FloorState Enum

ElevatorDisplay
- currentFloor: Number
- direction: Direction Enum
- floorState: FloorState Enum
- doorOpen: Boolean

ElevatorPanelButton
- pressed: Boolean

ElevatorPanel
- elevatorPassengers: Tenant[]
- currentFloor: Number

ElevatorControl
- currentFloor: Number
- direction: Direction Enum
- doorOpen: Boolean
- elevatorPassengers: Tenant[]
- moving: Boolean

PassengerRow
- name: String
- floor: number

PassengerList
- elevatorPassengers: Tenant[]

PendingRequestRow
- floor: number
- direction: Direction Enum
- requestTime: DateTime (Luxon)

PendingRequestsList
- pendingTenants: Tenant[]

Typescript Types

```
enum Direction {
    DOWN=0,
    UP,
    IDLE
}

interface Tenant {
   destinationFloor: number
   direction: Direction
   name: String
   requestTime: DateTime
   startingFloor: number
}

interface ElevatorState {
    currentFloor: number
    destinationFloor: number
    direction: Direction
    floorState: FloorState
    passengerQueue: Tenant[]
    requestQueue: Tenant[]
}

enum FloorState { - for determining what action to take next
    arriving: 0
    stopping: 1
    openDoors: 2
    closedDoors: 3
    leaving: 4
}
```

Redux Slices

Tenants Slice "tenants/"

PickUP(State): loads tenants at current floor matching direction & sorts

dropOff(state): unloads tenants at current floor w/ destfloor

addRequest(State, Tenant): Adds tenant to request Queue

Elevator Slice "elevator/"

nextFloor(state): increments/decrements floor by direction

updateDirection(state): Sets new direction according to State

updateDestination(state): Set new destination floor id according to State

stepFloorState(state): Moves floor State to next step

Misc Logic

```
ElevatorDisplayArrow (direction, floorState)
  if direction = 2; return flat line (yellow)
  else
      if FloorState = arriving or leaving → arrow is blinking
          if dir = 0, color is red, else green
      else its grey
          if dir = 0, arrow is down
```

Reducer Pseudocode

```
pickUp(State):
    newPassengers = State.requestQueue.findAll(where dir = State.dir
      &startingFloor = currentFloor)
    return {
        ...State,
        requestQueue: State.requestQueue.filter(not on newPassengers),
        PassengerQueue: [...newPassengers, ...State.PassengerQueue].SortbyDirection
    }
dropOff(state):
    return {
        ...State,
        PassengerQueue: State.PassengerQueue.filter(notdestfloor = State.currentFloor)
    }


addRequest(state, newTenant):
    return {
        ...State
        requestQueue: [...state.requestQueue, newTenant
    }


nextFloor(state):
    if State.direction = 2:
        return State
    return {
        ...State,
        floorState: 0,
        currentFloor: State.direction = 0 ? State.currentFloor-1 : State.currentFloor+1
    }
```

Finishing Touches

☒ Height Scaling for Passenger List & pending Requests (& make them Scrollable)

☒ Remove button interactability

☒ Add visual spawn timer

□ Add arrow animation

□ Better Door Icon/Animation