

# Pendulum Simulator Assignment

## About:

A pendulum simulator built for demonstration purposes

## How to run:

- Make sure you have RabbitMQ installed and started.
- **Setup Backend:** Navigate to the backend folder and launch 5 instances of index.js with node on the ports [8000, 7000, 5000, 4000, 3000]

```
node index.js 8000
node index.js 7000
node index.js 5000
node index.js 4000
node index.js 3000
```

- **Setup Frontend:** Navigate to the frontend folder and launch index.html
- You are now ready to go! When ready use the sliders to set a offset angle, a string length and mass for each pendulum and then set the wind factor and damping for the simulation.
- Hit set, this will create a pendulum instance with the set parameters and the information about its neighbours in each port.
- Use the start, pause and reset buttons to control the simulation.
- The export button, exports a JSON to the console in the bottom of the screen containing the current config.
- Upon collision of two pendulums, the simulation should stop in the backend and freeze in the frontend and restart after 5 seconds.

## Dependencies

- `express : ^4.17.1`
- `axios : ^0.21.1`
- `cors : ^2.8.5`
- `amqplib: ^0.6.0`

## REST API:

Our frontend uses `/setparams` to setup the simulation and uses `/start`, `/pause` and `/reset` to control the simulation from an UI. The frontend also periodically polls the `/angle` API to update the angle of the pendulum in the UI.

The `/position` is used to obtain the xy-coordinates of the neighbouring nodes that is used in collision detection function which is called with each update/time-step in the backend pendulum manager class during simulation.

## URLs:

- `/setparams` [POST] → accepts parameters for a pendulum object and URLs for its left and right neighbours and updates the PendulumManager's pendulum object and neighbour information. The URL returns a JSON response with the status and the pendulum object.
- `/start` [POST] → starts the simulation by calling the start method of the PendulumManager.
- `/pause` [POST] → pauses the simulation by calling the pause method of the PendulumManager.
- `/reset` [POST] → resets the simulation by calling the reset method of the PendulumManager.
- `/angle` [GET] → return a JSON response containing the current angle offset of the pendulum of the pendulum manager.
- `/position` [GET] → return a JSON response containing the x and y co-ordinates of the pendulum's bob of the pendulum manager.

## Guaranteed delivery communication implentation || AMQP Exchange Fanout

Using RabbitMQ a Publish/Subscribe Messaging Model is setup.


For each node: (using a custom class called CommandsQueue in commands.js)

- After establishing a connection with the rabbitmq client. An channel is created.
- On this channel a fanout exchange with the name "pendulum\_commands" is created.
- For sending messages, each node (publisher) can push messages to this exchange which will forward the messages to all the available queues. This is done using the .publish() method of the channel by providing the exchange name and message. sendCommand method of the CommandsQueue class does this for us.
- to listen (subscribe) for each node we create a queue for the channel and bind the queue with our exchange. Now we can consume the messages in the queue which were forwarded by the exchange by calling the .consume() method on the channel by providing the queue name.

This way we can push the "STOP" and "RESTART" commands to this exchange and all nodes receive it from their respective queues and perform a stop simulation upon receiving the "STOP" command and wait for 5s and push a "RESTART" to the exchange. Each PendulumManager keeps count of the number of "RESTART" messages received and finally when the count reach 5, the complete system is restarted.

### Classes:

#### Position Vector:

- Purpose: A class to represent a point/position and perform operations on it.
- Attributes:
  -  → x-coordinate

- `y` → y-coordinate
- `add()` → method to add two position vectors

### **Pendulum:**

- Purpose: A class to implement pendulum physics logic
- Attributes:
  - `origin` → Position vector of the origin
  - `stringLength` → length of the string attached to the pendulum bob.
  - `angularOffset` → initial offset angle of the pendulum.
  - `wind` → max-wind for simulation
  - `damping` → damping factor for the pendulum
  - `updatePosition()` → method to update the angularOffset by one time step.
  - `getPosition()` → method to calculate and return the position vector of the pendulum's bob.

### **Pendulum Manager:**

- Purpose: A class to wrap all controls regarding a pendulum simulation into one class.
- Attributes:
  - `commandqueue` → queue to push STOP and RESTART commands in guaranteed communication channel.
  - `simulation_key` → a identification key used to start, pause and reset simulations
  - `pendulum` → pendulum object attached to the manager class.
  - `restart_count` → number of restart messages received
  - `setParams()` → method to create a pendulum object with the received parameters along with the information about the neighbouring pendulums.

- `start()` → a method to start simulation by calling the `updatePosition` method of the pendulum periodically using `setInterval()` and then in each time step checking for collision between neighbors (using `checkcollision()`) and sending a "STOP" message in our commandqueue if there has been a collision.
- `checkcollision()` → method to check if the distance between two neighbours is acceptable and the two are not colliding.
- `pause()` → a method to pause the simulation by clearing the Interval that was set upon start.
- `reset()` → a method to reset the pendulum object.

### CommandsQueue:

- Purpose: A class to setup the RabbitMQ publish/subscribe model using channels, exchanges and queues.
- Attributes:
  - `exchange_name` → name of the exchange
  - `channel` → RabbitMQ channel
  - `init()` → a method to initialize our RabbitMQ messaging system, this method creates a channel, a fanout exchange in the channel, a queue in the channel and then binds the queue to the exchange to receive messages published to the exchange. The queue setup and binding is done in `.listen()` method the class.
  - `sendCommand()` → publishes message to the exchange.

### Notes: Pendulum Physics

- Wind is implemented, by introducing a variable that is added to the velocity of the pendulum bob.
- The mass of the pendulum does not affect its acceleration as it will always be acceleration due to gravity.

- However the effect of wind will reduce with increase in the mass of the pendulum. So to implement this logic, the affect of wind is reduced when mass is increased.
- In the frontend UI, increase in mass is shown with an increase in radius of the pendulum's bob. The fact that a larger radius would mean greater effect of wind is ignored as we are considering mass and not size. (so the larger size is just a representation of the mass and not of the size)

## **Notes: Frontend**

- The export button, outputs our configuration as a JSON object onto the console which is shown in the bottom of the screen.
- The fps (polling rate of the frontend) is variable and default is 5fps or a update every 200ms.
- In the Frontend, Instead of rendering a Bob with positions (x, y), the angle is used to rotate the div by the top by shifting the rotation origin.