



# Senior Design Team 15: Debugger and Visualizer for a Shared Sense of Time on Batteryless Sensor Networks

Adam Ford

Allan Juarez

Riley Thoma

Anthony Rosenhamer

Quentin Urbanowicz

Maksym Nakonechnyy

Advisor: Dr. Henry Duwe

Clients: Dr. Henry Duwe & Vishal Deep

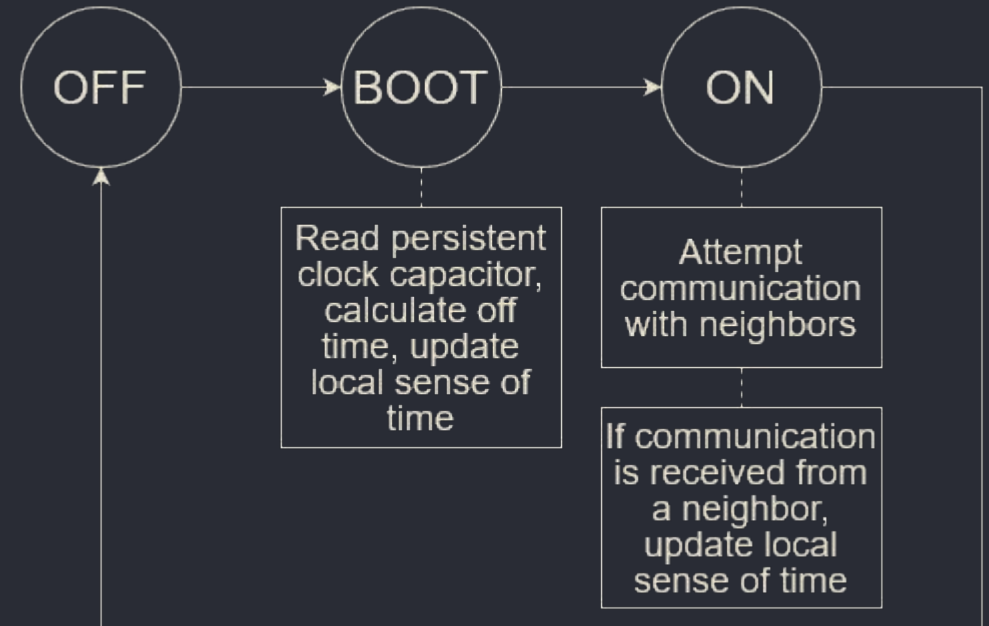
# Project Vision

## Need

- Networks of batteryless sensors

## Goal

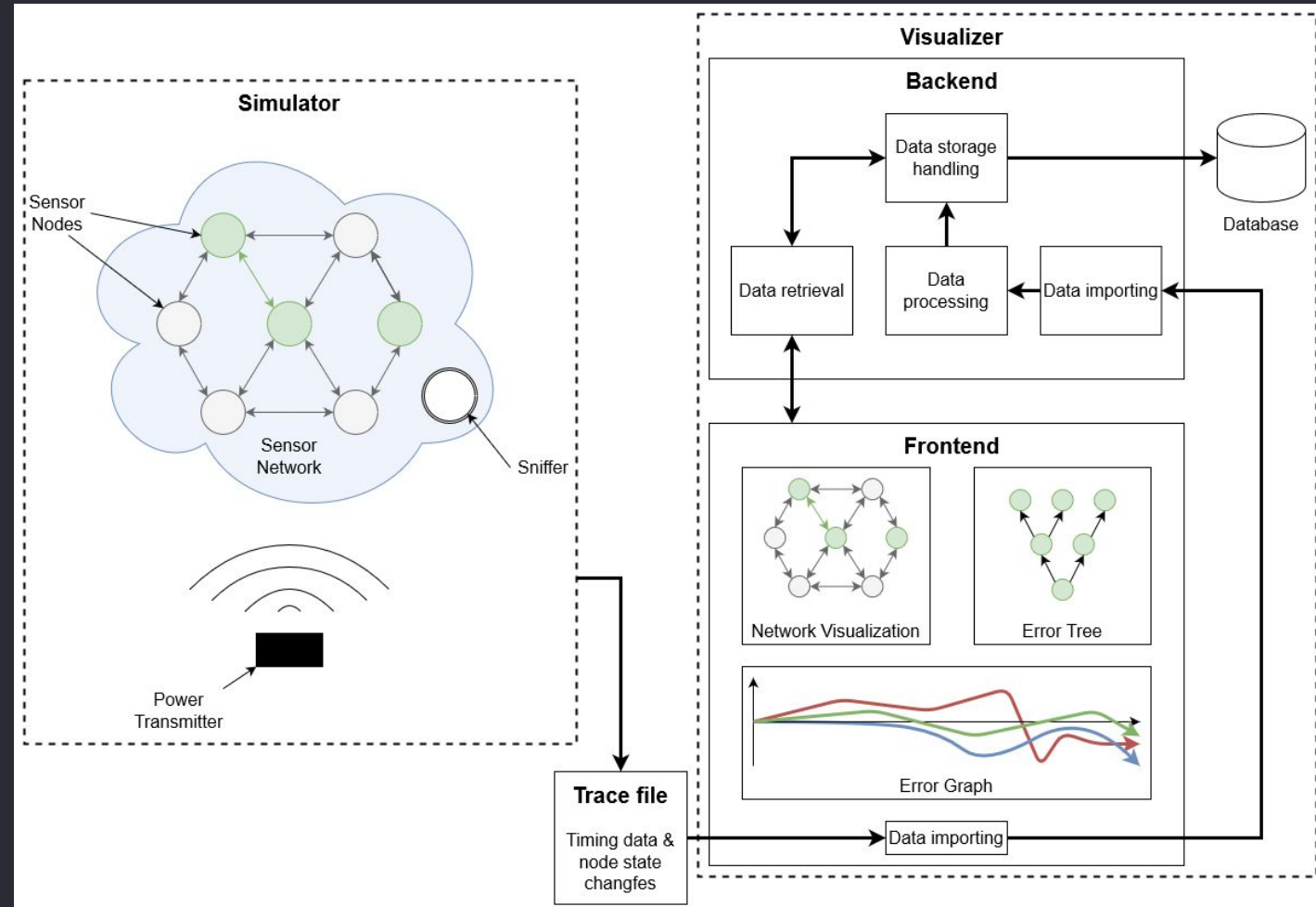
- Create software tools to simulate, visualize, and debug shared timekeeping in batteryless sensor networks



Lifecycle of a Node

# Conceptual Diagram

- Simulator
  - Models a sensor network
- Visualizer
  - Displays details about the sensor network



System Overview

# Functional Requirements

## Simulator

- Shall generate the data in the same format as real data.
- Shall produce on-time/off-time data from a user-selected energy model.

## Visualizer

- Shall visualize important time events selected by the user.
- Shall visualize the statistics of system communication.

## System-Wide

- Shall store trace file data.

# Non-Functional Requirements

## Simulator

- The simulator shall run natively in a Linux environment.

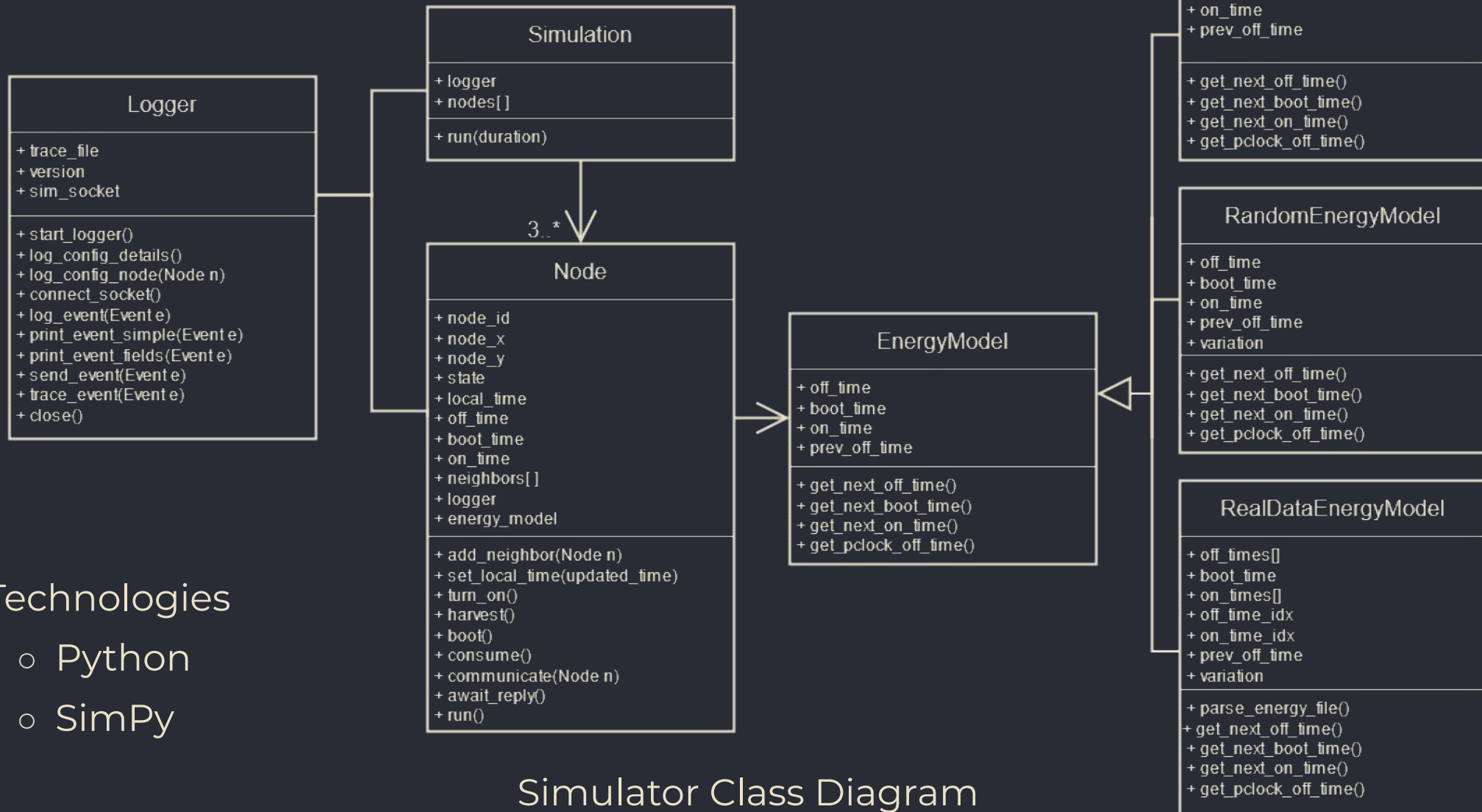
## Visualizer

- The visualizer shall be accessible from any OS
- The visualizer shall be implemented as a web application.

## System-wide

- The system shall be modular to allow for maintainability.

# System Design - Simulator



- Technologies

- Python
- SimPy

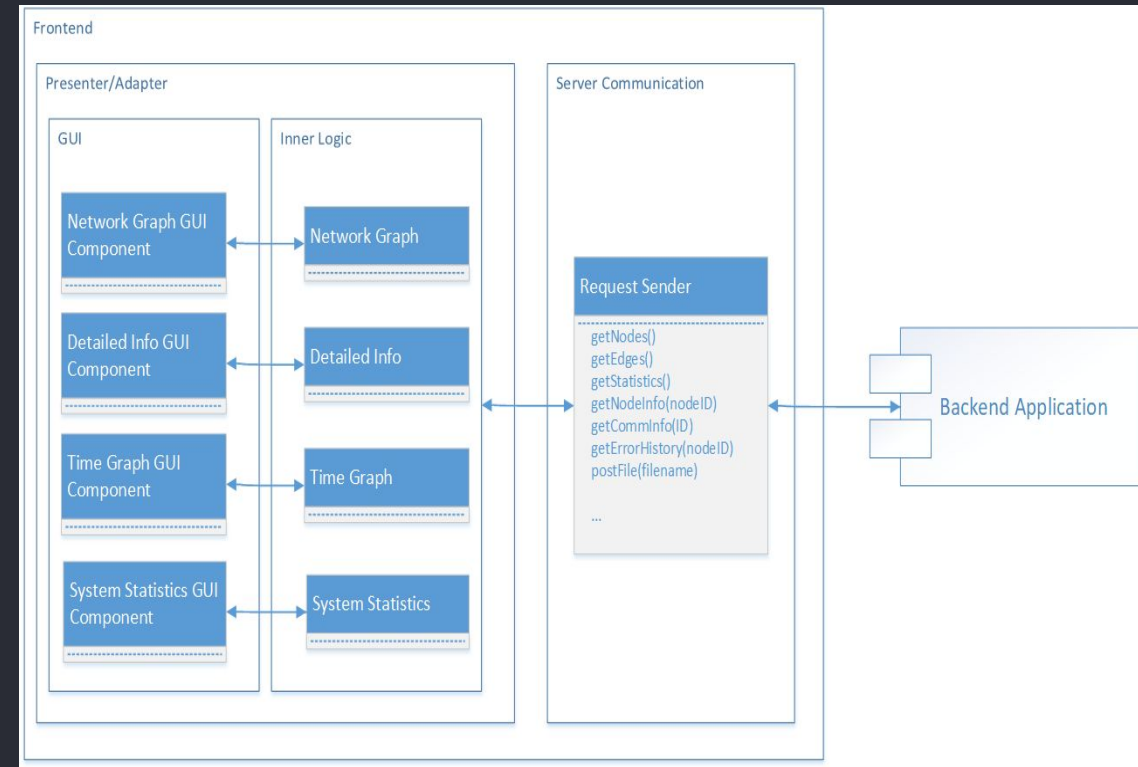
Simulator Class Diagram

# System Design - Backend

- Main Modules
  - Request Routing
  - Logic for Requests
  - Database queries
- Technologies
  - ExpressJS
  - MongoDB
  - Postman
- Design Updates
  - Socket Communication was dropped
  - Additional data manipulation was added for Error Trees and Graphs
  - Database structure for multiple simulations

# System Design - Frontend

- Consists of two logical components
  - Presenter/Adapter
    - Independent
    - Communicate via callbacks and EventBus
  - Server Communicator
- Communication with the backend via HTTP
- Technologies
  - HTML & CSS
  - React JavaScript
  - Jest & Selenium



Frontend Block Diagram



# Implementation - Simulator

- config.py: simulation is configured
  - Simulation duration and time step
  - Network structure
  - Node characteristics
- simulation.py: simulation is built and run
  - Configuration is applied
  - Node objects are defined
- node.py: nodes cycle between states
  - Gets times from energy model
  - Events are generated and logged

```
def run(self):  
    while True:  
        # Turns a node off and sits in the off state for the off-time duration  
        yield self.env.process(self.harvest())  
  
        # Performs boot up, updating the local time from the persistent clock  
        yield self.env.process(self.boot())  
  
        # Turns a node on, allowing communication to occur  
        self.turn_on()  
  
        # Yields to nodes that are booting or turning off  
        yield self.env.timeout(0)  
  
        # Handles communication with all neighbors  
        self.comm_time = 0  
        for n in self.neighbors:  
            if n not in self.communicated_with:  
                yield self.env.process(self.communicate(n))  
  
        # Keeps a node on for the remaining on-time duration  
        yield self.env.process(self.consume(self.comm_time))
```

Node Class run() Function

# Implementation - Backend

- Trace File Consumption/Upload
  - Input: 0x124F80030101010102020...
  - Output:

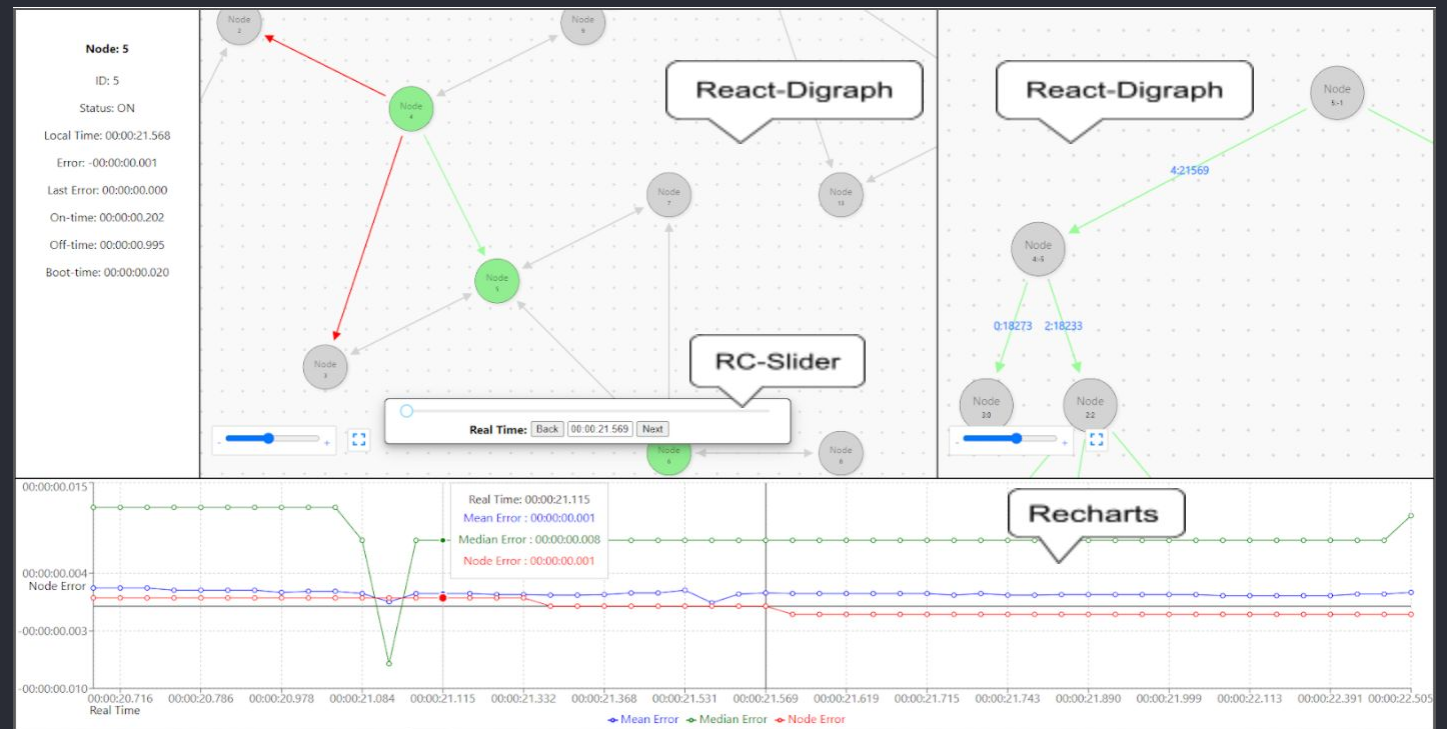
```
realTime: 1017
  nodes: Array
    0: Object
      id: 1
      localTime: 0
      onTime: 0
      offTime: 1017
      bootTime: 0
      error: -1017
      lastError: 0
      selfError: 0
      status: 0
    1: Object
    2: Object
    3: Object
    4: Object
    5: Object
    6: Object
    7: Object
  events: Array
    0: Object
      senderId: 2
      destId: 1
      didSucceed: false
    1: Object
```

- General Endpoints
  - Handle the multiple simulations
  - Query then Serve the data persisted in MongoDB
- Tree Creation
  - Algorithm that produces the error tree based on previous communications
  - Decided on a nested “recursive” format

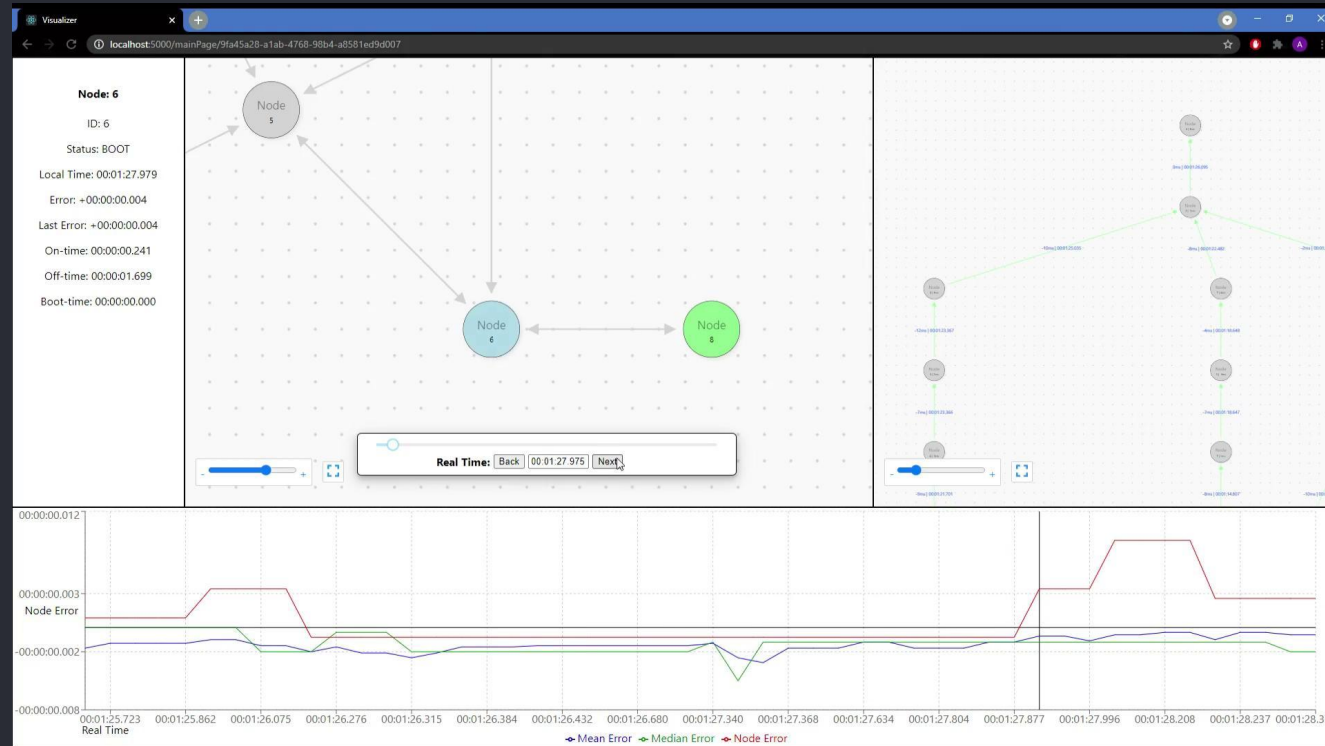
# Implementation - Frontend

- Two main pages developed in ReactJS with HTML & CSS
- Publisher - subscriber pattern used for GUI updates
- Buchheim algorithm for drawing rooted trees

## Libraries Utilized



# Demo



# Testing - Simulator

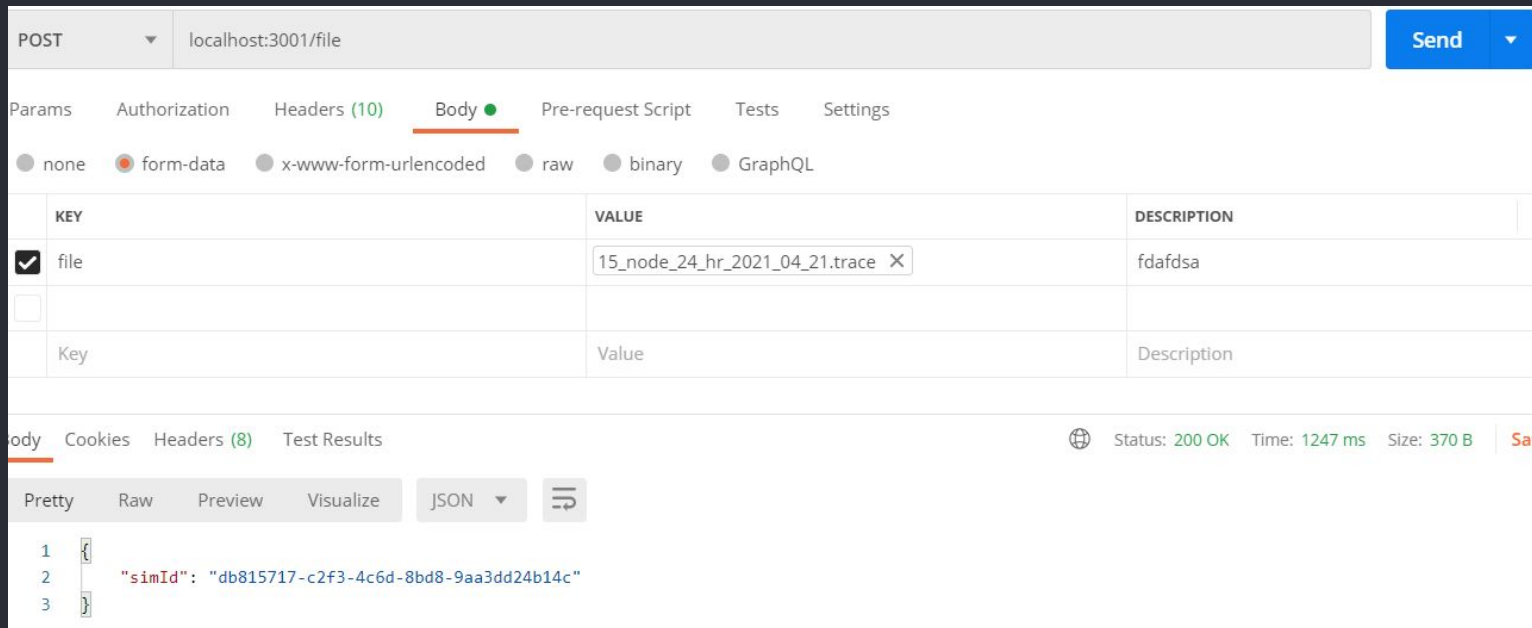
- Component unit testing
  - Mock backend used to test backend interface
- Simulation performance testing:

Duration (simulated time)	10 minutes	2 hours	24 hours
Actual run time	5.86 sec	1 min, 9.94 sec	14 min, 9.05 sec
Seconds of run time per simulated second	0.00977	0.00971	0.00983
Seconds of run time per simulated second per node	<b>0.00122</b>	<b>0.00121</b>	<b>0.00123</b>

8-Node Simulation Performance

# Testing - Backend

- Used Postman to make HTTP calls on our endpoints to see what results we would get
- Compared the event text file (provided by the simulator) to what we have stored in the database.



The screenshot shows a Postman interface for a POST request to `localhost:3001/file`. The request body is set to `form-data` and contains a single entry with the key `file` and the value `15_node_24_hr_2021_04_21.trace`. The response status is `200 OK` with a time of `1247 ms` and a size of `370 B`. The response body is displayed in JSON format:

```
1 {
2   "simId": "db815717-c2f3-4c6d-8bd8-9aa3dd24b14c"
3 }
```

# Testing - Frontend

- Custom backend
- Manual testing
- Jest unit test suite
- Selenium GUI test suite

Project: SD Visualizer\*

Executing test\_suite\_0111

test\_launch\_sim

test\_move\_to\_next\_event

test\_move\_to\_previous\_event

test\_slider\_change

test\_edge\_click

test\_no\_node\_selected

test\_node\_click

test\_select\_node\_and\_display\_error\_tree

test\_select\_node\_and\_move\_time

test\_successful\_communication

test\_failed\_communication

test\_error\_tree\_graphing

test\_reaching\_edge

Runs: 15 Failures: 0

localhost:5000

	Command	Target	Value
1	open	http://localhost:5000/	
2	set window size	1064x815	
3	click	css=td:nth-child(3) > input	
4	click	css=input:nth-child(5)	
5	click	id=node-2	
6	assert text	css=p:nth-child(3)	Status: BOOT
7	click	css=input:nth-child(5)	
8	click	css=input:nth-child(5)	
9	click	css=input:nth-child(5)	
10	assert text	css=p:nth-child(3)	Status: ON

Command: open

Target: http://localhost:5000/

Value:

Description:



Thank you!

Questions?