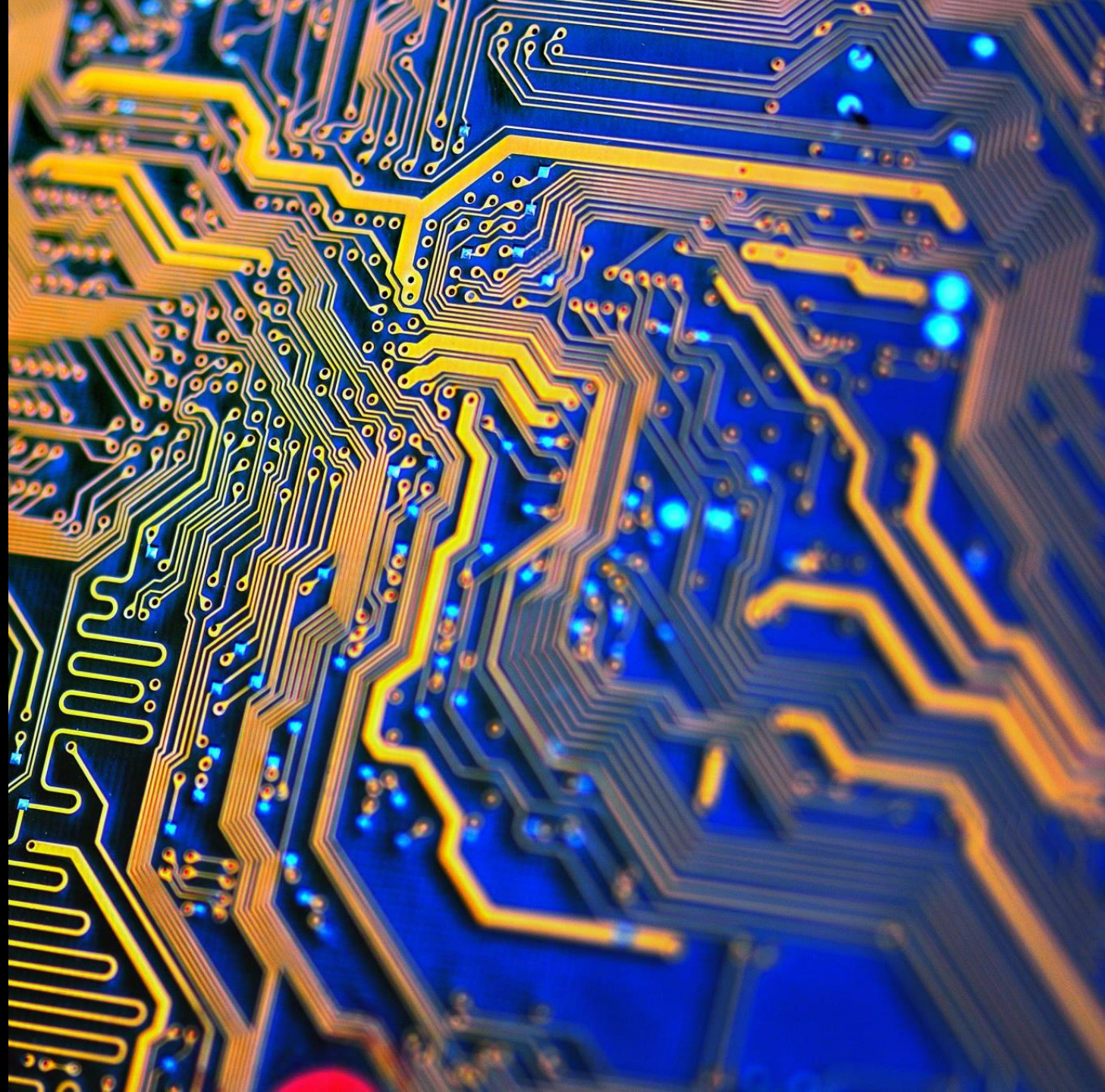
TERRARIUM IOT-HYPERLEDGER PROJECT

- Fawaz Malik

  #101461582
- Namit Mani

  #101383808
- Aniket Srivastava #101469899
- BADARUDDIN KHUHRO

  #101467663
- HEMANTH KUMAR POTHURI #101464127



WHY BLOCKCHAIN

- Transparency
- Security
- Efficiency
- Trust
- No Tokens needed
- Scalability Benefits
- Trackability

OUR CONSENSUS MECHANISM

- Proof of Elapsed Time
  - Cheap (economical)
  - Token-less (helps in live implementation)
  - Randomized (safer)

USERS INVOLVED

- Potential Users of the Blockchain Network:
  - Shop-owners
  - Pet-Owners
  - Small IoT Device Manufacturer

USE CASE #1

- Purchase Scenario:
  - customer buys pet and terrarium tank from shopowner. Transaction stored on blockchain Hyperledger fabric with button click on front end application
  - Shop owner sells terrarium tank and a pet with IoT device to monitor health and safety status to consumer
  - Consumer can test the IoT at the shop using Start tank button on front end

BENEFITS

- Two-fold:
  - 1 For New Customers :- Ease of Ownership, ability to choose the level of interaction in the pet's life and allows for a better "new parent" experience.
  - 2 For Old Customers :- The ability to have an even more immersive ownership experience. An easier ownership experience as well as e-managed abilities that

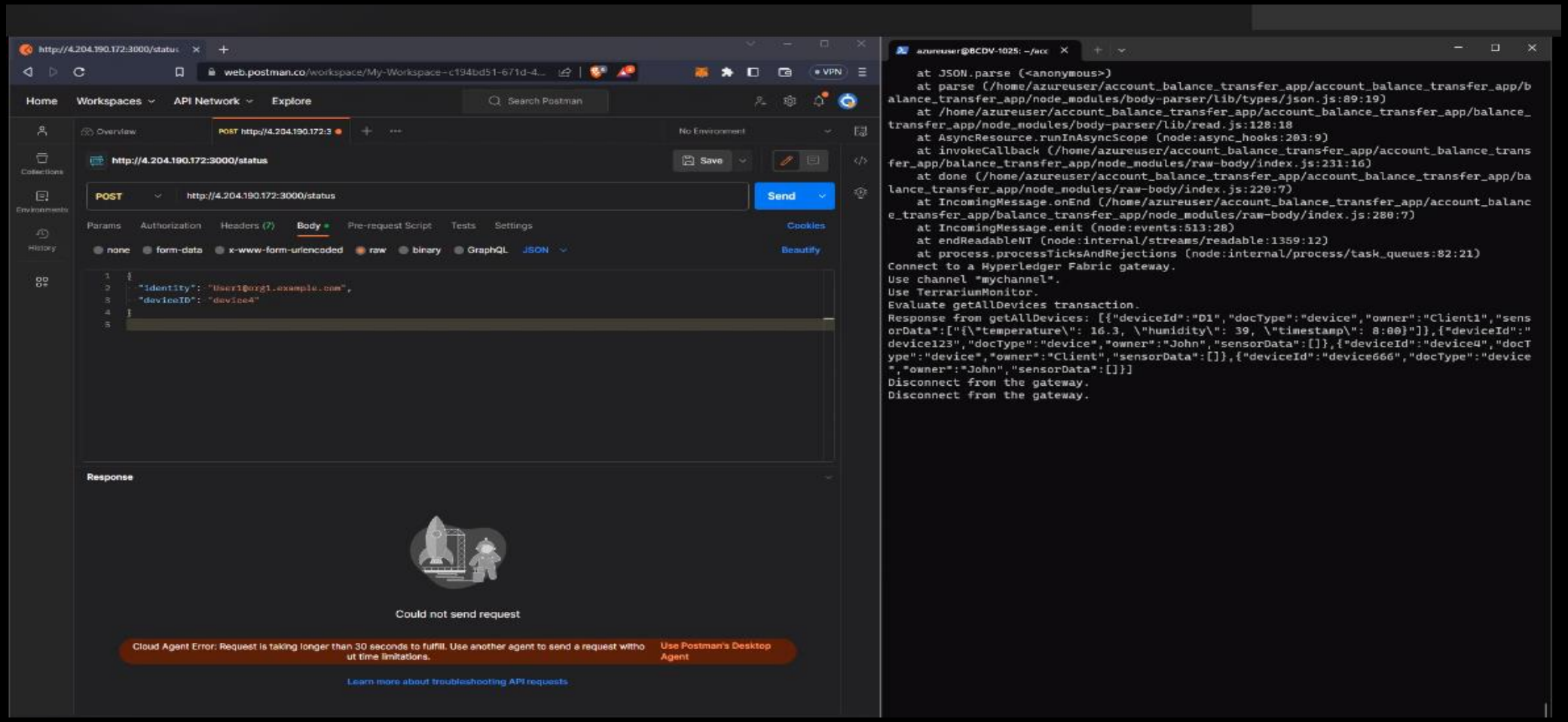
ERN STACK

- Dev Stack- Express.js, React.js, Node.js
- Front end user interface consists of Nodejs and react components
- Back end consists of Express talking back door with our Hyperledger fabric chaincode

OUR SMART CONTRACT

```javascript
/*
 * SPDX-License-Identifier: Apache-2.0
 */

'use strict';

const { Contract } = require('fabric-contract-api');

class PetTerrariumContract extends Contract {
    async initLedger(ctx) {
        // Initialize any initial data or setup here
    }

    async buyPet(ctx, petId, ownerId, petName, purchaseDate) {
        const pet = {
            ownerId,
            petName,
            purchaseDate,
            status: 'Purchased',
            tankId: '',
            isTankStarted: false,
        };

        await ctx.stub.putState(petId, Buffer.from(JSON.stringify(pet)));
    }

    async buyTank(ctx, petId, tankId, purchaseDate) {
        const petBytes = await ctx.stub.getState(petId);
        if (!petBytes || petBytes.length === 0) {
            throw new Error(`Pet with ID ${petId} does not exist.`);
        }

        const pet = JSON.parse(petBytes.toString());
        if (pet.status !== 'Purchased') {
            throw new Error(`Pet with ID ${petId} is not in a valid state for buying a tank.`);
        }

        pet.tankId = tankId;
        pet.purchaseDate = purchaseDate;
        await ctx.stub.putState(petId, Buffer.from(JSON.stringify(pet)));
    }

    async startTank(ctx, petId) {
        const petBytes = await ctx.stub.getState(petId);
        if (!petBytes || petBytes.length === 0) {
            throw new Error(`Pet with ID ${petId} does not exist.`);
        }

        const pet = JSON.parse(petBytes.toString());
        if (pet.status !== 'Purchased' || !pet.tankId) {
            throw new Error(`Pet with ID ${petId} is not in a valid state for starting the tank.`);
        }

        pet.isTankStarted = true;
        await ctx.stub.putState(petId, Buffer.from(JSON.stringify(pet)));
    }

    async getPet(ctx, petId) {
        const petBytes = await ctx.stub.getState(petId);
        if (!petBytes || petBytes.length === 0) {
            throw new Error(`Pet with ID ${petId} does not exist.`);
        }

        return petBytes.toString();
    }
}

module.exports = PetTerrariumContract;
```

CHAINCODE INSTALLATION & DEPLOYMENT



LOGICAL ILLUSTRATION OF OUR CONTRACT