```
const axios = require('axios');
const express = require('express');

const app = express();
const PORT = 8000;

// Window storage (to maintain the last 10 numbers)
let windowState = [1, 4, 6, 8, 9, 3];
const windowSize = 10;

const api_urls = {
    p: 'http://testserver.com/api/prime',
    f: 'http://testserver.com/api/fibonacci',
    e: 'http://testserver.com/api/even',
    r: 'http://testserver.com/api/random',
};

// Utility function to fetch numbers with timeout
const fetchNumber = async (type) => {
    try {
        const response = await axios.get(api_urls[type], { timeout: 500 });
        return response.data.numbers || [];
    } catch (error) {
        console.error(error);
        return [];
    }
```

```
};

// Helper function to check if a number is prime
const isPrime = (num) => {
    if (num < 2) return false;
    for (let i = 2; i <= Math.sqrt(num); i++) {
        if (num % i === 0) return false;
    }
    return true;
};

// Helper function to check if a number is in Fibonacci sequence
const isFibonacci = (num, a = 0, b = 1) => {
    while (b < num) [a, b] = [b, a + b];
    return b === num || num === 0;
};

// API Route with Query Parameter Filtering
app.get('/numbers/:type', async (req, res) => {
    const { type } = req.params;
    if (!['p', 'f', 'e', 'r'].includes(type)) {
        return res.status(400).json({ error: "Invalid Number Type" });
    }

    // Store previous state before updating
    let previousState = [...windowState];
```

```
    // Fetch new numbers from external API
    let newNumbers = await fetchNumber(type);

    // Filter based on type
    if (type === "e") newNumbers = newNumbers.filter(n => n % 2 === 0);
// Even numbers
    if (type === "p") newNumbers = newNumbers.filter(isPrime); // Prime
numbers
    if (type === "f") newNumbers = newNumbers.filter(isFibonacci); //
Fibonacci numbers
    if(type==="r")
newNumbers=newNumbers.filter(generateRandomNumbers)
    // Append unique numbers to window state
    newNumbers.forEach(n => {
        if (!windowState.includes(n)) {
            windowState.push(n);
        }
    });

    // Maintain only last 10 numbers
    while (windowState.length > windowSize) {
        windowState.shift();
    }

    // Set current state
    let windowCurrState = [...windowState];
```

```
    // Apply the same filtering on previous and current state
    let filteredPrevState = previousState;
    let filteredCurrState = windowCurrState;
    if (type === "e") {
        filteredPrevState = previousState.filter(n => n % 2 === 0);
        filteredCurrState = windowCurrState.filter(n => n % 2 === 0);
    } else if (type === "p") {
        filteredPrevState = previousState.filter(isPrime);
        filteredCurrState = windowCurrState.filter(isPrime);
    } else if (type === "f") {
        filteredPrevState = previousState.filter(isFibonacci);
        filteredCurrState = windowCurrState.filter(isFibonacci);
    }
    else if(type=="r"){
        const generateRandomNumbers = async (count, delayTime) => {
            let randomNumbers = [];
            for (let i = 0; i < count; i++) {
                await new Promise(resolve => setTimeout(resolve, delayTime));
// Introduce delay
                randomNumbers.push(Math.floor(Math.random() * 100)); //
Random number between 0-99
            }
            return randomNumbers;
        };
    }
```

```
    // Create numbers array with unique values from filtered previous and
current states
    let numbers = [...new Set([...filteredPrevState, ...filteredCurrState,
...newNumbers])];


    // Calculate average
    const avg = numbers.length
        ? (numbers.reduce((sum, num) => sum + num, 0) /
numbers.length).toFixed(2)
        : 0;



    res.json({
        windowPrevState: previousState,
        windowCurrState: windowCurrState,
        numbers: numbers,
        avg: parseFloat(avg),
    });
});

app.listen(PORT, () => {
    console.log(`Server running on ${PORT}`);
});
```