National College of Ireland

Configuration Manual

MSc Research Project
Msc in Cloud Computing

Praneeth Raghava Vadrevu
Student ID: 23211946

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Praneeth Raghava Vadrevu
Student ID:	23211946
Programme:	Msc in Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Praneeth Raghava Vadrevu
23211946

1 Introduction

This document elaborates the tools, technologies and frameworks used for the implementation of the project. It also describes the steps carried out in the implementation of the research project, "A Scalable Blockchain Framework for Access Control in Cloud Environments".

2 System Configuration

2.1 Software Specification

- Visual Studio Code installed on a systems operating on MacOS.
- Installed node to further install Vite(React), Express and Hardhat frameworks for interface design, smart contracts and server development and testing. Aws
- Additionally k6 a load testing tool along with PostMan an API testing software were installed.
- Added MetaMask as a browser extension which is a Web 3.0 wallet for handling tokens.

2.2 Hardware Specification

- MacBook Air M2, 256GB SSD, 8GB Ram

2.3 Steps to run the framework

2.3.1 Rpc Provider

Need to setup a project on Alchemy which is an Rpc Provider to get the API key as depcited on 1. Alchemy (n.d.)

2.3.2 Local Network

- **Deploying the Smart Contract**: Open the file "WhisperNet" from code artefacts on VSC similar to 2 . Navigate to hardhat using "cd hardhat" command. Solidity (n.d.)

  Run "npx hardhat compile" to compile the smart contract, refer to 3.

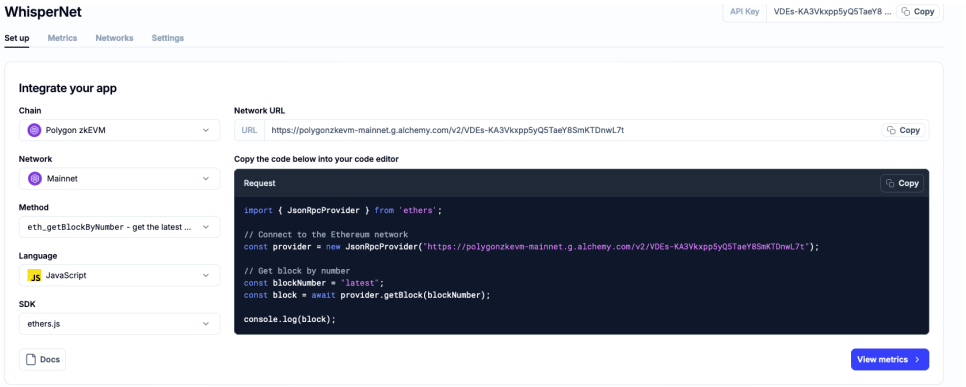

Figure 1: Alchemy.

To run the node server "npx hardhat node" and open a new terminal as depicted in 4.

Now run "npx hardhat run –network localhost scripts/deploy.cjs" to deploy the smart contract on local 5.

"npx hardhat test" runs the test files, refer to 6 and 7.

- **Express Server**: To use the express server, the smart contract has to be deployed to a blockchain, in this case polygonZKEVM. Express (n.d.)

  Run "npm install" to install all the dependencies.

  First step is to add the private address of your web 3.0 wallet to the .env file next to PRIVATE_ADDRESS = "". MetaMask (n.d.)

  Then run "npx hardhat run –network polygonZk scripts/deploy.cjs" which will deploy the smart contract, make sure there are enough tokens on the polygonZKEvm network like in 9.

  A successful deployment will print the contract address in the terminal, which needs to be used in the .env next to CONTRACT_ADDRESS="".

  Now run "node index.cjs" command to start the express/node server.

  The API end points can be tested using PostMan ex:login, register. Refer to 10 and Postman (n.d.)

- **React**: Navigate to client folder using terminal "cd client".

  Run "npm install" to install all the dependencies. Vite (n.d.)

  In the .env file of client, change the END_POINT to the ideal port where express server is up and running. The client will make API calls using this.

  With everything ready run the vite server using command "npm run dev" and upon running the url you will see something like 11.

3 Running Tests

To perform the required tasks, first of all, install k6 from its official website – k6. io k6 (n.d.) and create a JavaScript test. The script in this case uses the "http" module

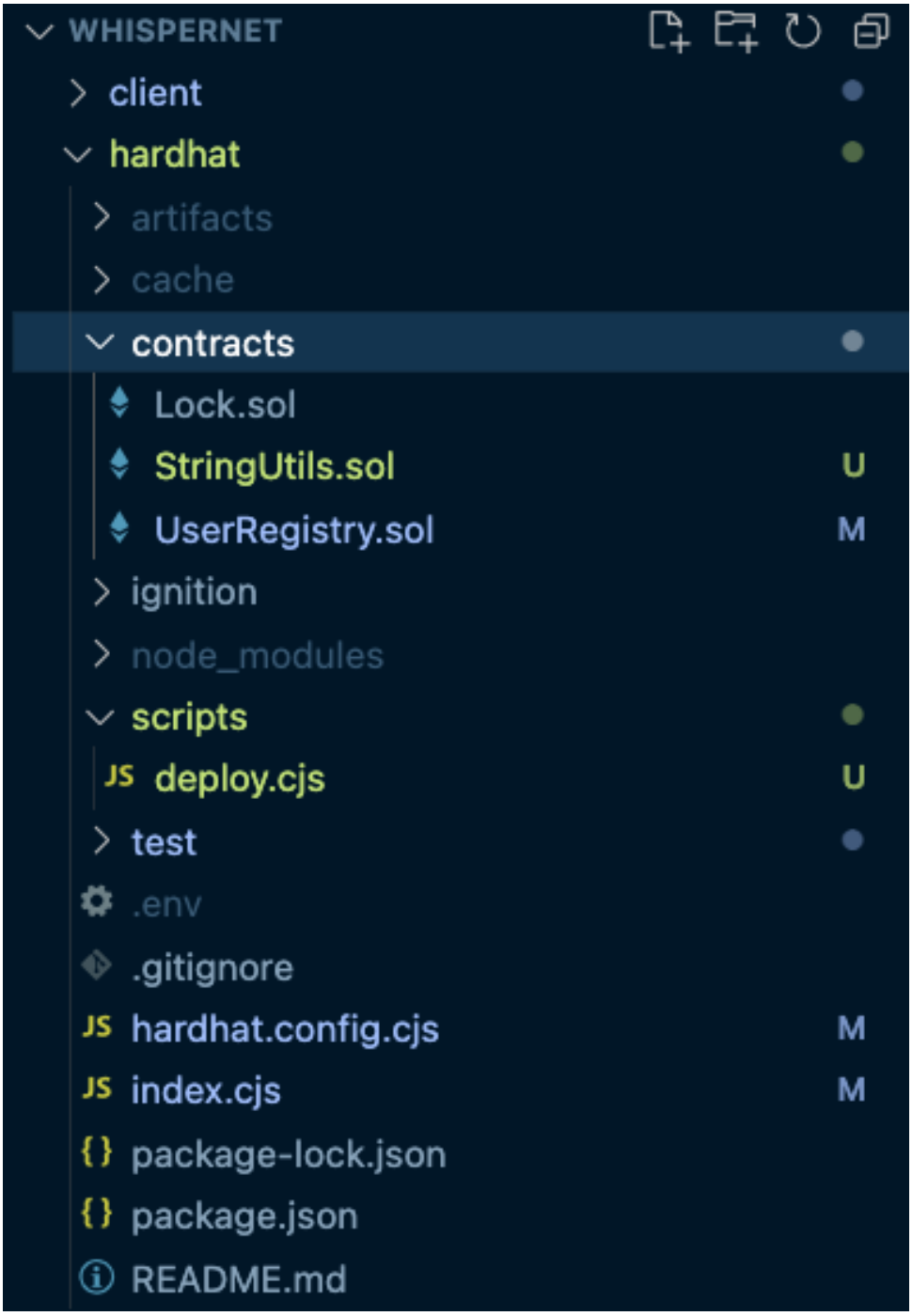

Figure 2: Folder Structure.

```
praneethraghav@Praneeths-MacBook-Air hardhat % npx hardhat compile
Compiled 1 Solidity file successfully (evm target: paris).
```

Figure 3: Hardhat Compile.

to issue HTTP requests and the "check" function to verify the responses. Virtual users or VUs can be configured with load test options, which include defining the stages of loading, ramping up, sustaining and ramping down of the VUs. For instance, you can mimic that the users are growing over 30 seconds, remain the same for one minute and

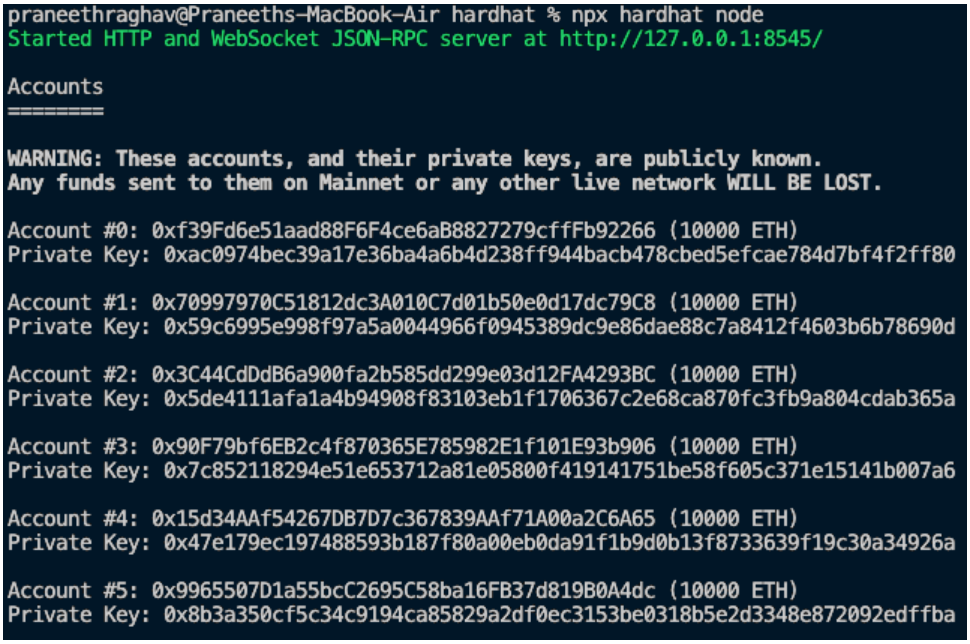

Figure 4: Hardhat Node.

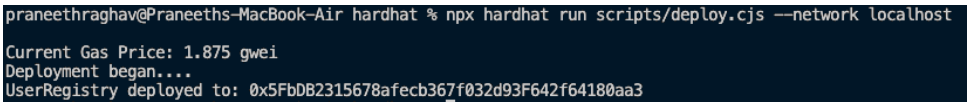

Figure 5: Hardhat Deploy.

then reduce. Performance goals can be established in the form of thresholds to ensure that certain performance standards are met regarding such factors as response time for instance, response time for 95% of the requests should not be more than 2 seconds. To start the test, execute the command k6 run in the terminal and monitor the performance of the the system production-ready in configurations real-time include and the get ability the to results tear for down request resources duration during and teardown failure such rate as among the others. use Some of of teardown() or integrating with tools like Grafana for monitoring.

References

Alchemy (n.d.). Alchemy - Blockchain Development Platform, `https://www.alchemy.com/`. Accessed: 2024-12-12.

Express (n.d.). Express - Fast, unopinionated, minimalist web framework for Node.js, `https://expressjs.com/`. Accessed: 2024-12-12.

k6 (n.d.). k6 Load Testing Tool, `https://k6.io/`. Accessed: 2024-12-12.

MetaMask (n.d.). MetaMask, `https://metamask.io/`. Accessed: 2024-12-12.

Postman (n.d.). Postman - The Collaboration Platform for API Development, `https://www.postman.com/`. Accessed: 2024-12-12.

```
it("should log in the user with correct credentials and listen for events", async function () {

    const emailHash = ethers.keccak256(ethers.toUtf8Bytes("raghav@gmail.com"));
    const passwordHash = ethers.keccak256(ethers.toUtf8Bytes("12345"));

    // Retrieve the current nonce
    const nonce = await contract.getNonce(emailHash);
    console.log("Retrieved nonce:", nonce.toString());

    // Listen for the "NonceUpdated" event
    let eventEmitted = false;

    contract.on("UserLoggedIn", (loggedEmailHash, newNonce, loggedUserIdHash) => {
        console.log("UserLog event triggered:", {
            emailHash: loggedEmailHash,
            newNonce: newNonce.toString(),
            userIdHash: loggedUserIdHash,
        });

        // Validate the event data
        expect(loggedEmailHash).to.equal(emailHash);
        expect(newNonce.toString()).to.equal((BigInt(nonce) + BigInt(1)).toString());
        expect(loggedUserIdHash).to.equal(userIdHash);

        eventEmitted = true;
    });

    // Call the login function
    await contract.connect(user1).login(emailHash, passwordHash, nonce);

    // Wait for the event to be emitted
    await new Promise((resolve) => setTimeout(resolve, 1000));
    expect(eventEmitted).to.be.true;

    // Verify the user's last login time
    const userDetails = await contract.getUser(userIdHash);
    console.log("Last login time:", userDetails.lastLogin.toString());
    expect(userDetails.lastLogin).to.be.gt(0); // Ensure lastLogin was updated
});
```

Figure 6: Login test.

```
praneethraghav@Praneeths-MacBook-Air hardhat % npx hardhat test


  UserRegistryOptimized - Login
    ✔ should map emailHash to userIdHash correctly
User IDs: Result(2) [
  '0x42d359ce00a5ec64abfa8f756a224808be066e840f6843cfb970d65ef9910846',
  '0xeb7f32ef93e47ba759b5f2c9cce12ee76655ff71d9e3bd5946fce69a169c4f43'
]
User Exists: Result(2) [ true, true ]
Nonces: [ '1', '1' ]
Email to User IDs: Result(2) [
  '0x42d359ce00a5ec64abfa8f756a224808be066e840f6843cfb970d65ef9910846',
  '0xeb7f32ef93e47ba759b5f2c9cce12ee76655ff71d9e3bd5946fce69a169c4f43'
]
    ✔ should retrieve all registered records
Retrieved nonce: 1
```

Figure 7: Test Result.

```
praneethraghav@Praneeths-MacBook-Air hardhat % npx hardhat run scripts/deploy.cjs --network polygonZK

Current Gas Price: 0.257 gwei
Deployment began....
UserRegistry deployed to: 0x84B8207ef94794D1207459ec0313bc5646840aAB
```

Figure 8: Deploying to Polygon.

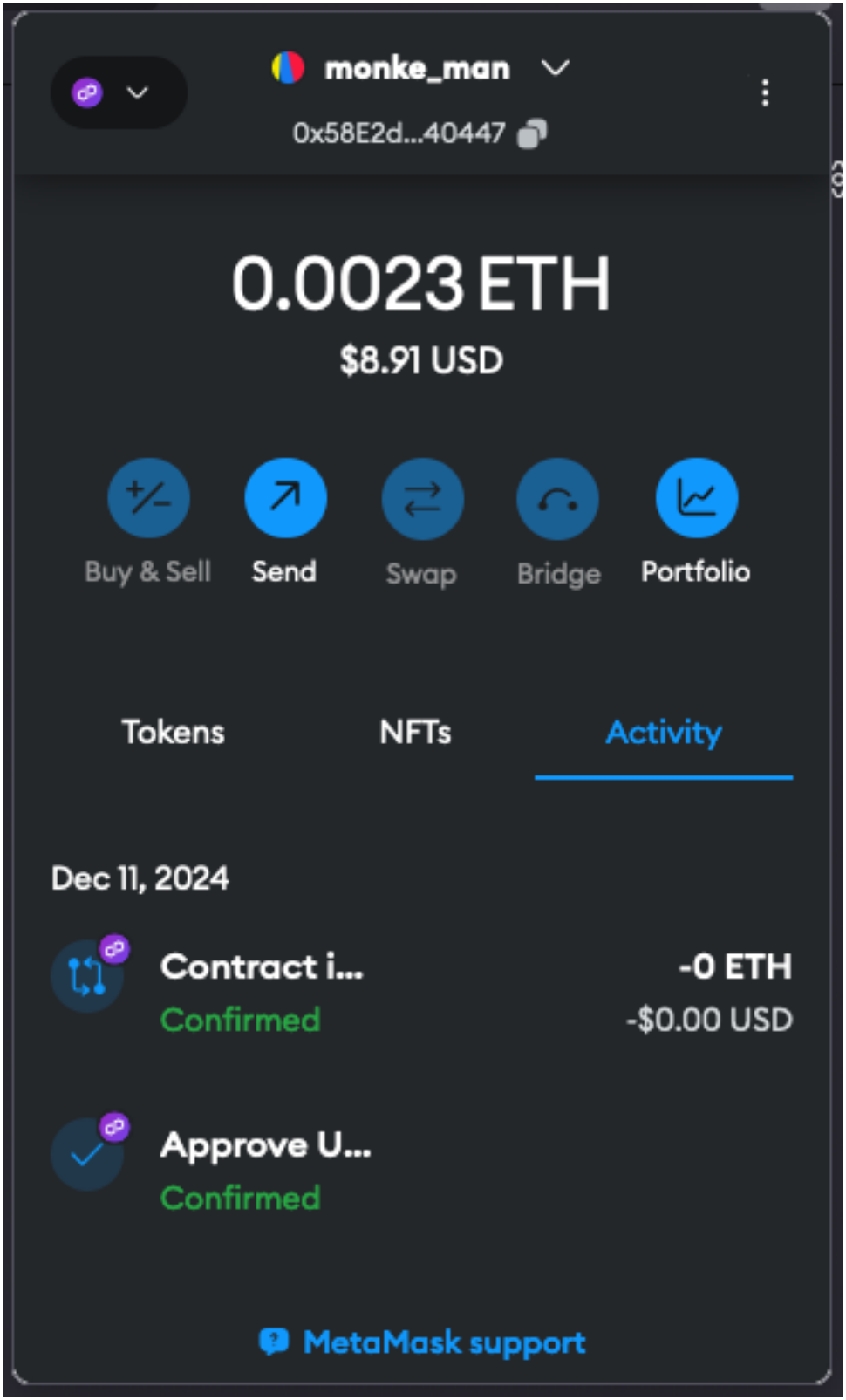

Figure 9: MetaMask.

Solidity (n.d.). Solidity Documentation, `https://soliditylang.org/`. Accessed: 2024-12-12.

Vite (n.d.). Vite - Next Generation Frontend Tooling, `https://vitejs.dev/`. Accessed:

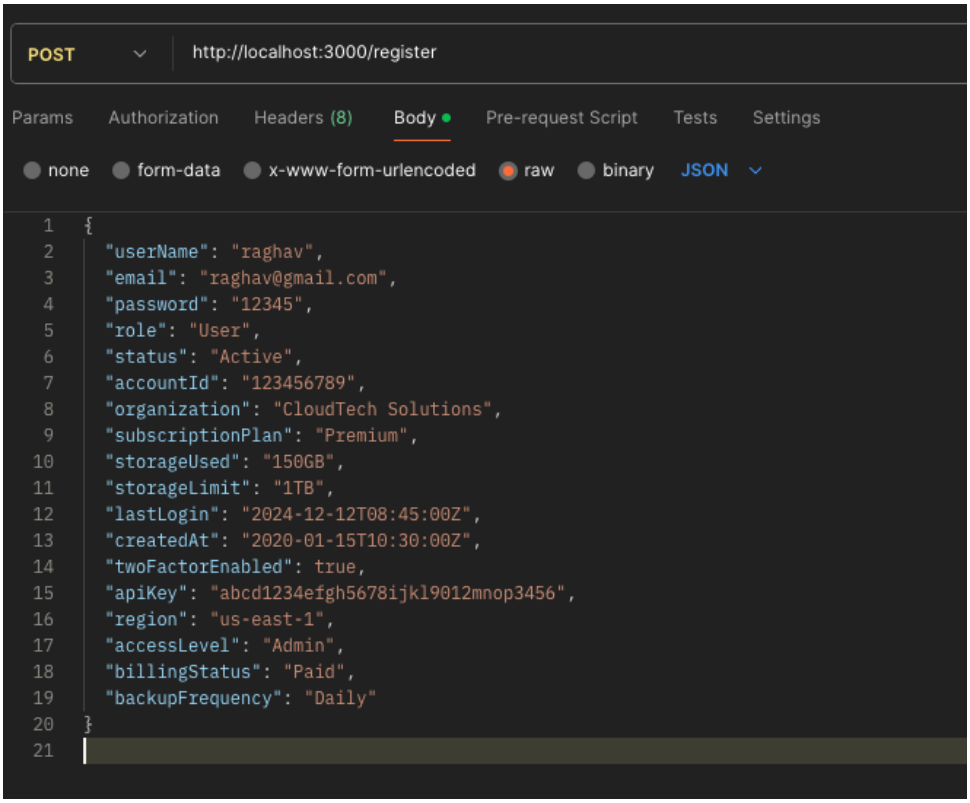

Figure 10: PostMan.

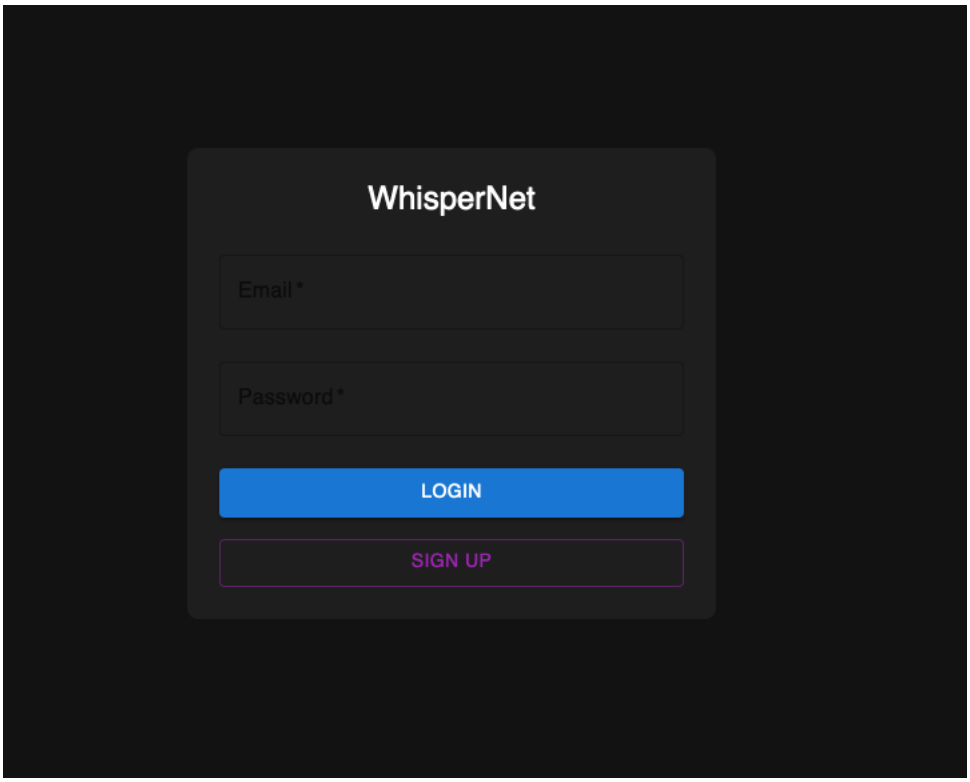

Figure 11: Login.

2024-12-12.

```
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Counter } from 'k6/metrics';

export const options = {
  stages: [
    { duration: '30s', target: 10 }, // Ramp up to 10 users over 30 seconds
    { duration: '1m', target: 10 },  // Stay at 10 users for 1 minute
    { duration: '30s', target: 0 },  // Ramp down to 0 users over 30 seconds
  ],
  thresholds: {
    http_req_duration: ['p(95)<2000'], // 95% of requests must complete below 2000ms
    'http_req_failed{scenario:default}': ['rate<0.01'], // Fail rate < 1%
  },
};

const BASE_URL = 'http://localhost:3000'; // Replace with your actual API URL
const loginEndpoint = '/login';
const email = 'testuser@example.com';
const password = 'testpassword123';

const failureCount = new Counter('failed_requests');

export default function () {
  const payload = JSON.stringify({
    email: email,
    password: password,
  });

  const params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  const res = http.post(`${BASE_URL}${loginEndpoint}`, payload, params);

  // Validate response
  const checks = check(res, {
    'is status 200': (r) => r.status === 200,
    'is success true': (r) => JSON.parse(r.body).success === true,
    'response time < 1s': (r) => r.timings.duration < 1000,
  });
```

Figure 12: K6.