### Biography:

How old are you?

* 29

Have you used JavaScript before?

* Yes

Have you ever written asynchronous code? (async, await, promise, then, …)

* Yes

From a scale of 1-10 rate your understanding of how asynchronous code works in JavaScript?

* 6

### Reading the code:

Please read the code below and try to fully understand its functionality.

# Code.js: Credit: This code is written by ChatGPT 3.5

| const fs = require('fs').promises; const fetch = require('node-fetch');  const API\_KEY = 'your\_api\_key'; *// Replace with your actual API key* const CITIES = ['New York', 'London', 'Tokyo']; *// List of cities to fetch weather for* const FILE\_PATH = 'example.txt'; *// Path to the file to read*  async function fetchWeather(city) {  try {  const response = await fetch(`http://api.openweathermap.org/weather?q=${city}&appid=${API\_KEY}`);  const data = await response.json();  return { city, weather: data.weather[0].description };  } catch (error) {  console.error(`Error fetching weather for ${city}:`, error);  throw error;  } } async function readFile(filePath) {  try {  const data = await fs.readFile(filePath, 'utf8');  return data;  } catch (error) {  console.error('Error reading file:', error);  throw error;  } } async function main() {  try {  console.time('totalTime');   *// Perform IO operation (reading from file) asynchronously*  const fileData = await readFile(FILE\_PATH);  console.log('File contents:', fileData);   *// Perform multiple API requests concurrently using Promise.all*  const weatherPromises = CITIES.map(city => fetchWeather(city));  const weatherData = await Promise.all(weatherPromises);   *// Log weather data for each city*  weatherData.forEach(data => {  console.log(`Current weather in ${data.city}: ${data.weather}`);  });   } catch (error) {  console.error('An error occurred:', error);  } }  main(); |
| --- |

Did you fully understand the functionality of the provided code?

* Yes

Can you estimate the order of function executions? If so, please write them.

Tbh I don't really know what order of function executions are… :’) but anyways:

* Console.time
* ReadFile
* Console.log
* CITIES.map
* Fetch weather x3
* Promise.all
* Weather data.forEach
* Console.log x3
* If error anywhere, console.error

How much time do you expect each async function in the code will use? (in milliseconds or % of the whole program)

* fetchWeather(): this probably depends on a lot of factors like the API efficiency, internet speed, etc?
* readFile(): this probably depends how big the file is?
* main(): 100%

How much CPU/memory resource do you expect each async function in the code will use? (can be in % of the whole program)

* fetchWeather(): memory usage depends how much information is returned from the API?
* readFile(): memory usage depends in how big the file is
* main(): if the resources are dynamically allocated, then maybe it could be close to 100%, since those do seem to be much more potentially resource heavy than the other parts of the code?

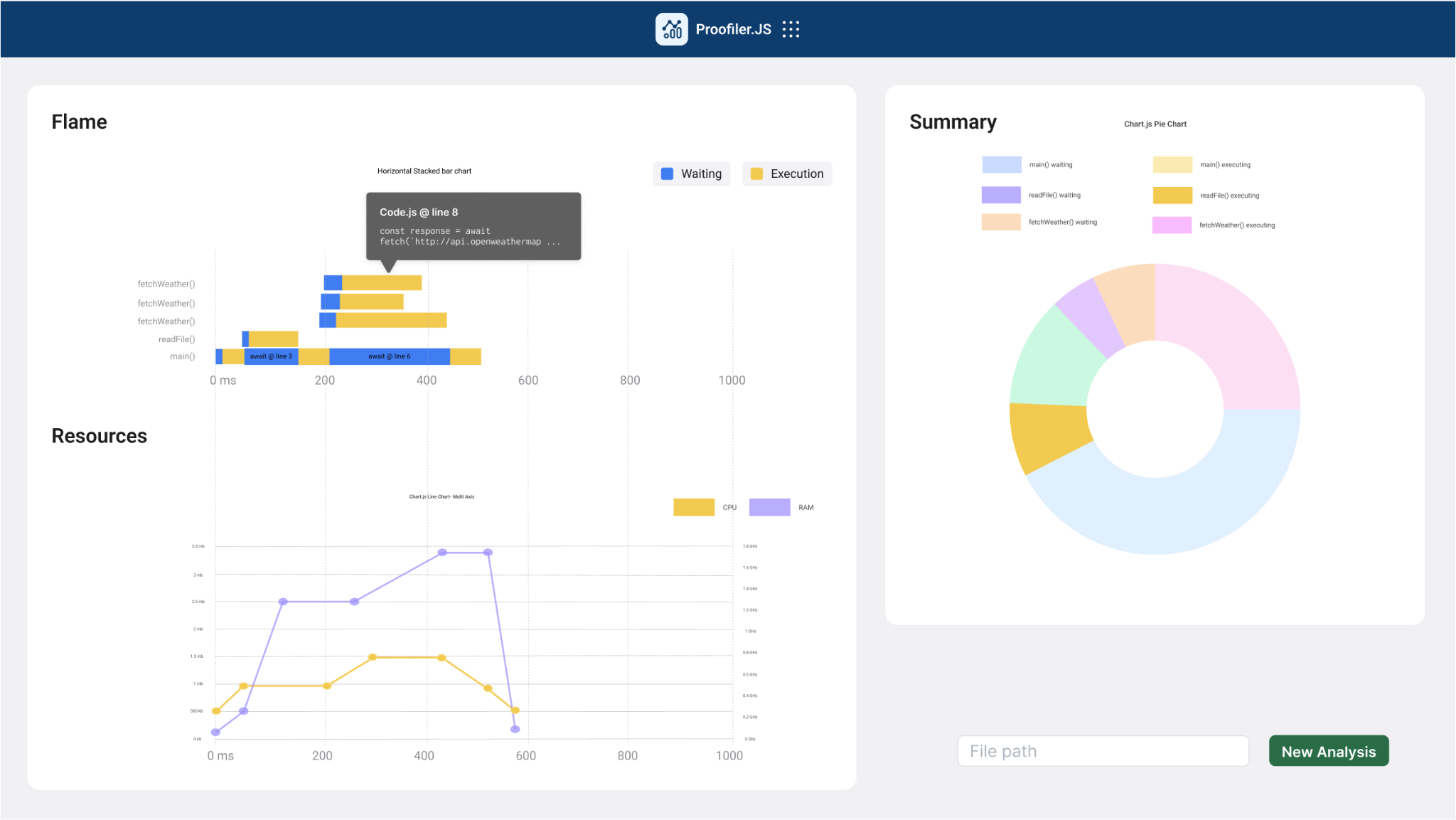
### 

### 

***Please do not proceed until you answer the questions on the previous page.***

### Visualization:

The following image is a screenshot of a profiler specialized for asynchronous functions. In this instance, we are representing the code you examined earlier. Please take a look and try to understand the graphs.



How long did it take you to understand the relation between pieces of the graph and the code?

* 1 min?

By looking at the profiler, did you learn anything new about the code performance or order of execution which you didn’t expect when inspecting the code?

* That the waiting times and executing times look like they can be quite different is interesting

In terms of usefulness for debugging an async code, give a score of 1-10 for each part of the profiler:

I think this depends on what exactly I'm debugging. E.g. the flame graph would be useful for debugging things related to the order of execution, the flame graph and the summary graph would be useful for debugging things related to code speed, and the resources graph would be useful for debugging issues related to resource usage.

1. Flame: 9
2. Resources: 9
3. Summary: 9

In terms of intuitiveness of graphs, give a score of 1-10 for each part of the profiler:

1. Flame: 10
2. Resources: 9
3. Summary: 8 (I think one of the colors might be off? :D)

Imagine you were debugging an asynchronous code. What other pieces of info do you think would have been useful in the profiler?

* Variable information would be cool! Like what the variables were at the different execution steps in the flame graph
* Something that combines the resources graph with the flame graph might be useful in debugging what executing functions might be causing high resource usage

Finally, from a scale of 1-10 how would you overall rate the profiler?

* 9 looks great!