**Technical Solution Approach**

Contents

[1 Introduction 2](#_Toc127885736)

[1.1 About this document 2](#_Toc127885737)

[1.1.1 Purpose & Scope of the document 2](#_Toc127885738)

[2 Component Design 2](#_Toc127885739)

[2.1 Component Design Diagram 2](#_Toc127885740)

[2.1.1 Overall Workflow 2](#_Toc127885741)

[2.1.2 Low level Design 2](#_Toc127885742)

[3 Technology & Frameworks to be used 3](#_Toc127885743)

[4 Solution Approach 3](#_Toc127885744)

# Introduction

## About this document

### Purpose & Scope of the document

Application health has evolved into a critical component of every organization. An unstable (unhealthy) application causes significant revenue loss, erodes client trust, and has a negative impact on business. To avoid such situations, we should keep track of the status of our applications on a frequent basis.

End Users can utilize Health Checks in.NET Core to constantly check the health of the program, be proactive in discovering any problems, and make the necessary adjustments rather than waiting for the end user or system to tell them that there is a problem with the application. When end users build Health Checks, they are better qualified to diagnose issues with application architecture because they can quickly determine which service or dependency is wrong. This enables end users to create extremely fine, precise checks for services. Putting health checks in place assists users in understanding what a healthy state of an application looks like because it is possible that it is still functioning but in a degraded form that is difficult to detect by using it.

# Component Design

## Component Design Diagram

Application health checks are when you specify "healthy" criteria for your application's monitoring metrics and run regular checks to ensure the system is working as intended.

With these health checks, we can be warned of outages before customers, reducing the impact of the outage by resolving it or telling the client about the current outage with time to up, resulting in confidence.

Monitoring is dependent on the application's complexity and criticality. A basic program may just need its dependencies monitored once per day, whereas a vital application may need to be monitored as frequently as feasible. We can also provide a status page where developers may examine the results and add features to inform them of problems.

**WORKING:**

When an application generates a request to the server. The server sends a response to the application. When the server sends the response, it will be captured by the health check component, it will display the result if it is healthy or not. The request that is generated will fetch out all the responses that are collected from the server, and filter out the necessary request and display it in the UI. Clearly, we can't send all requests to the UI, so we will specify the request and filter it out. The specified services that are required will be displayed. if the server doesn't send the response of the service, it will be marked as unhealthy. Also, if the server responds to the request and it stays idle, it will be marked as "diagonized."

### Overall Workflow

### 

Fig 1.1

Sample Sequence diagram for the health refer Fig 1.1:

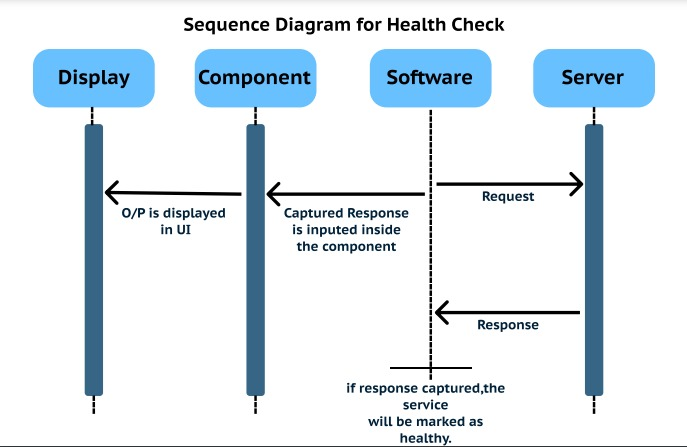


Fig 1.2

### Low level Design

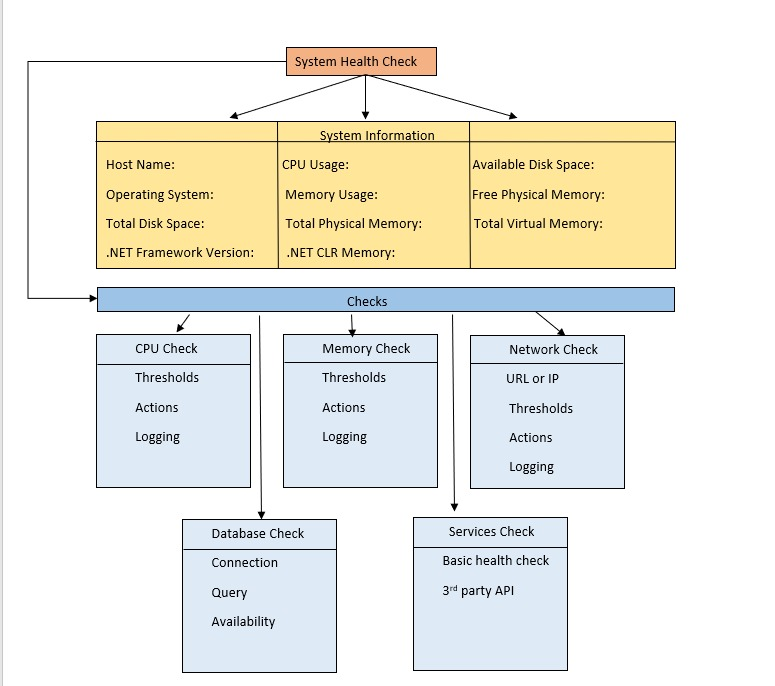


Fig 1.3

# Technology & Frameworks to be used

C# Programming language

.NET Framework

AspNetCore.HealthChecks (Open source Library).

RESTful API

# Solution Approach

1. Based on the design diagram, we might consider constructing a webpage to display the service checks.

2. The design pattern uses an open source library to fetch the services activity and respond in JSON format.

3. The UI component will be in charge of presenting the service's health and response time in conjunction with the log generated.

4. The UI component will use the information stored in JSON format to display on the webpage.

5. This component will then call the service, check its status, and send us the report.

6. This component will be developed in.Net and C-Sharp as a self-contained module.