Health Checks in .Net Core

Component Specification Document

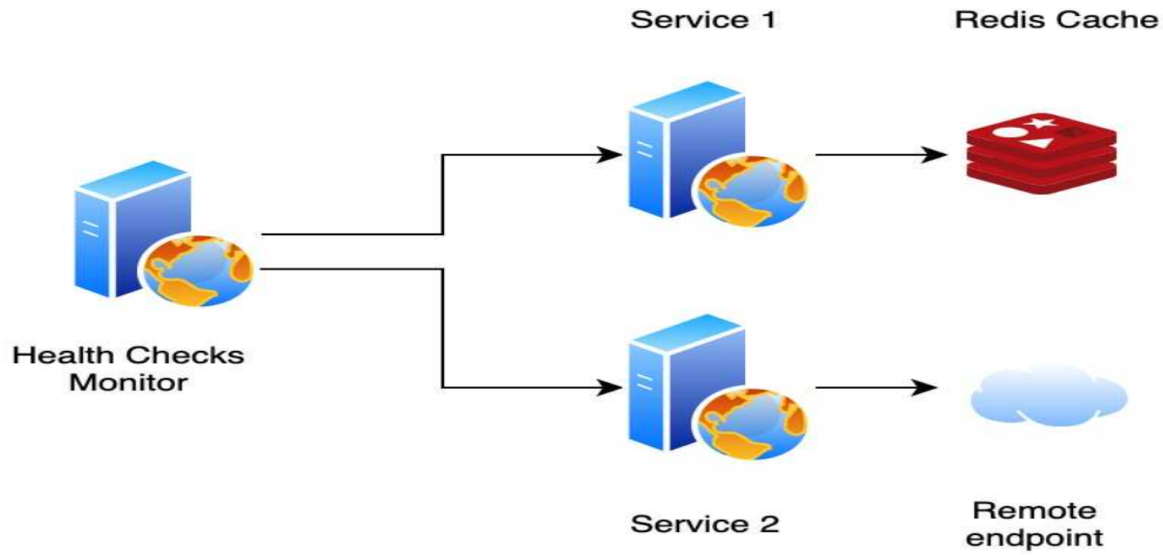
V1.0

	Prepared By / Last Updated By	Reviewed By	Approved By
Name			
Role			
Signature			
Date			

Table of Contents

1.0Abstract

Health Checks in .NET Core helps the End Users to constantly monitor the health of the program, be proactive in identifying any difficulties, and make the appropriate adjustments rather than waiting on the end user or system to alert them there is a problem with the application. End users are better equipped to diagnose issues with application architecture when they design Health Checks because than can rapidly determine which service or dependency is malfunctioning. This allows end users to establish incredibly granular, precise checks for services. Having health checks in place helps users understand what a healthy condition of an application looks like because it's possible that it's still running but in a degraded form that is difficult to discover by using it.



1.1 Technology/Framework for Development

- .NET Framework 4.5.1
- Visual Studio 2019
- .NET Core 6.0

This is a detailed informative template for .NET use case but this can be also implemented with spring boot Java as well as with python to create the health services and make it available for the UI Dashboard.

2.0Business Scenario

This component can be usable for the End Users in the below scenarios

- Checking a dependence whose status is out of the end user's control and that is external to the end user (like 3rd party web services and rest services).
- Examining the condition of application data dependencies like Cache Servers and SQL Server databases.
- Examining the endpoints on user own site for timing and performance.
- Verifying the settings for those that may be challenging to determine in a multi-environment system.
- Exposing health checks as HTTP endpoints. Different real-time monitoring scenarios can be specified for health check endpoints.
- Checking the functionality of the physical server's memory, disc, and other resources.

Component Features

The component should have the following features

- Applications should be provisioned with exposed endpoints that can be accessed by outside tools which will assist in confirming that the services and applications are operating correctly.
- Provision to provide information, specifically on the component portions of an end-user application's functionality.
- Provision to include health checks for external services, databases, caches, and even end users' custom health-checks.
- Provision for the End Users to understand what is happening inside the website to find problems, recognize security issues, and locate optimization opportunities.
- Provision to indicate when an app's start-up processes are complete, and the app is ready to start handling requests.
- Provision to expose endpoints on websites which can provide rapid diagnostic information on the status of end users' applications while also executing several pre-written tests.
- Provision for End User to configure and personalize several different sorts of Health Checks.

3.0Sample Approach

Health check in application can be done in following ways

- Create health checks
- Create the stub application for which we need to monitor the heath checks.
- Register health check services
- Use Health Checks Routing
- Require authorization
- Enable Cross-Origin Requests (CORS)

Options for health check behaviour should include filtering health check results, customizing the HTTP status code, suppressing cache headers, and customizing the output.

The fundamental configuration invokes the Health Checks Middleware to register health check services and to respond with a health response at a URL endpoint. No health checks are recorded by default to test any given dependency or subsystem. If the app can reply to the health endpoint URL, it is deemed healthy. Health Status is sent to the client as an unencrypted response by the default response writer.

4.0Health Check Sample Test Scenario

Below is the Health Check Sample Test Scenario

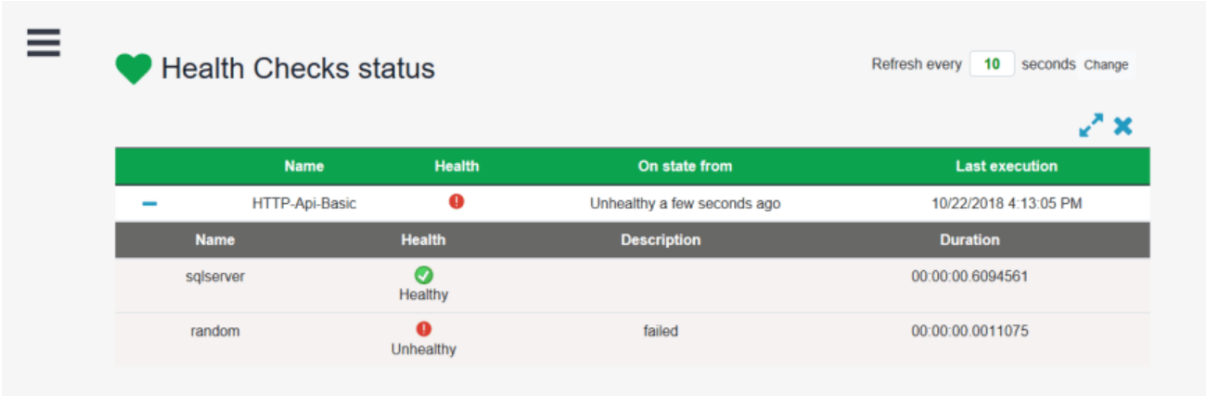
- Run the application and the database in a local/ Docker container
- Navigate to localhost:<port>/alive: end user should get Healthy as result
- Stop the database container and refresh the page: end user should still get Healthy as result
- Navigate to localhost:<port>/dB context: end user should get Unhealthy as result
- Restart the database container and refresh the page: you should get Healthy as result

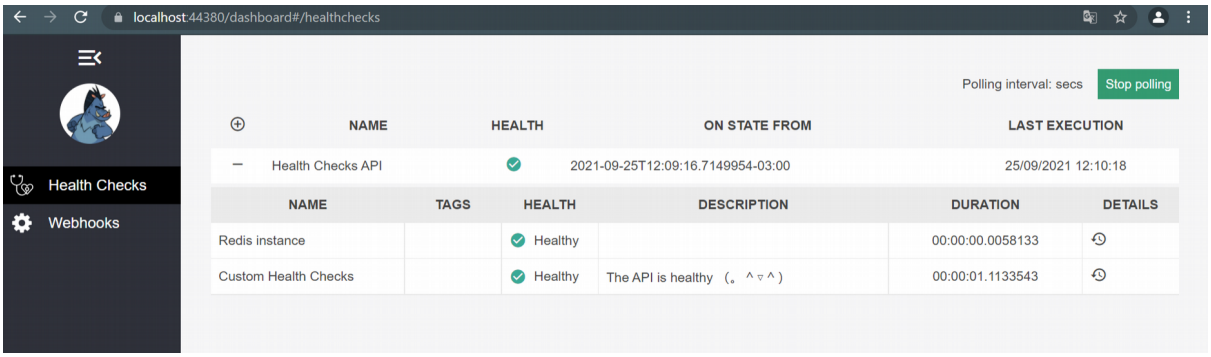
5.0Health Check Test Screen Logs

Overall Screen

A user interface (UI) would be considerably more useful than JSON for displaying the health check service's output. End user may view the health check output as a web interface as illustrated below.

The health status of our application should hopefully change every five seconds because of the random value obtained from this UI's refresh rate.



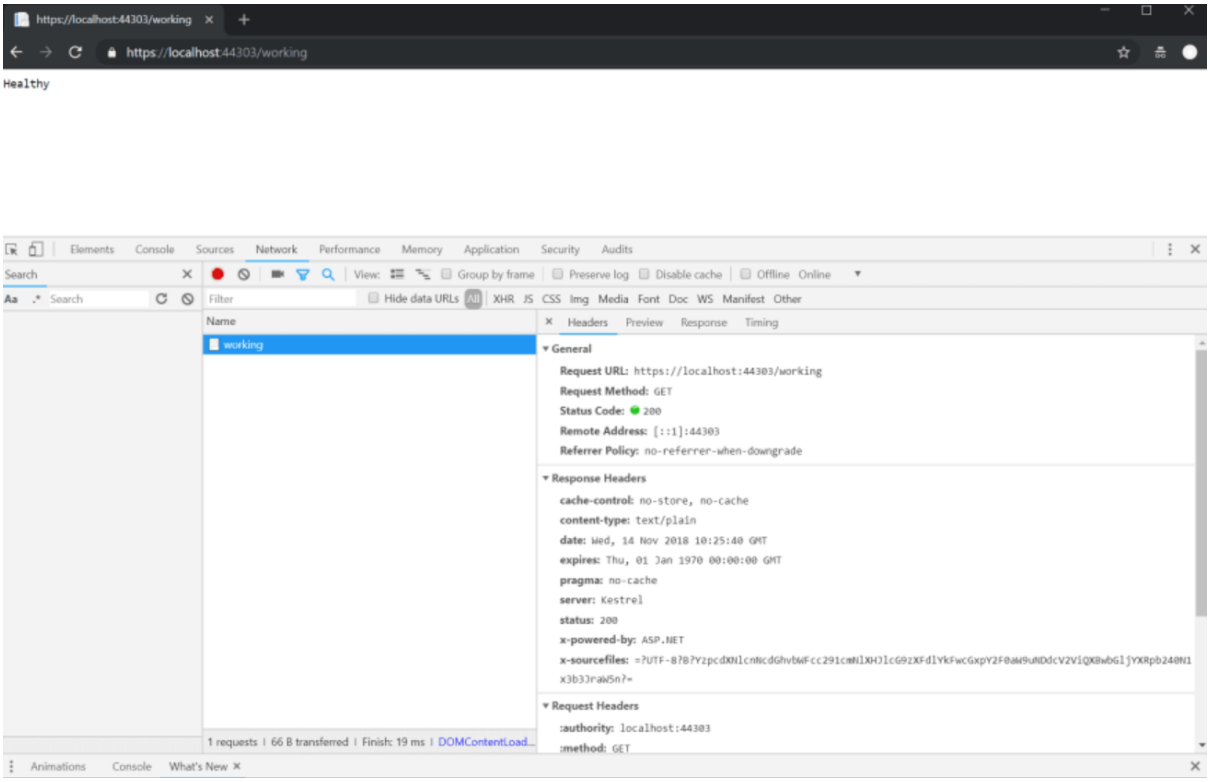


Additionally, by repeatedly running health check services using a variety of storage options, such as SQL, In Memory, and audit trials, it can save the outputs on storage.

Healthy Message Screen

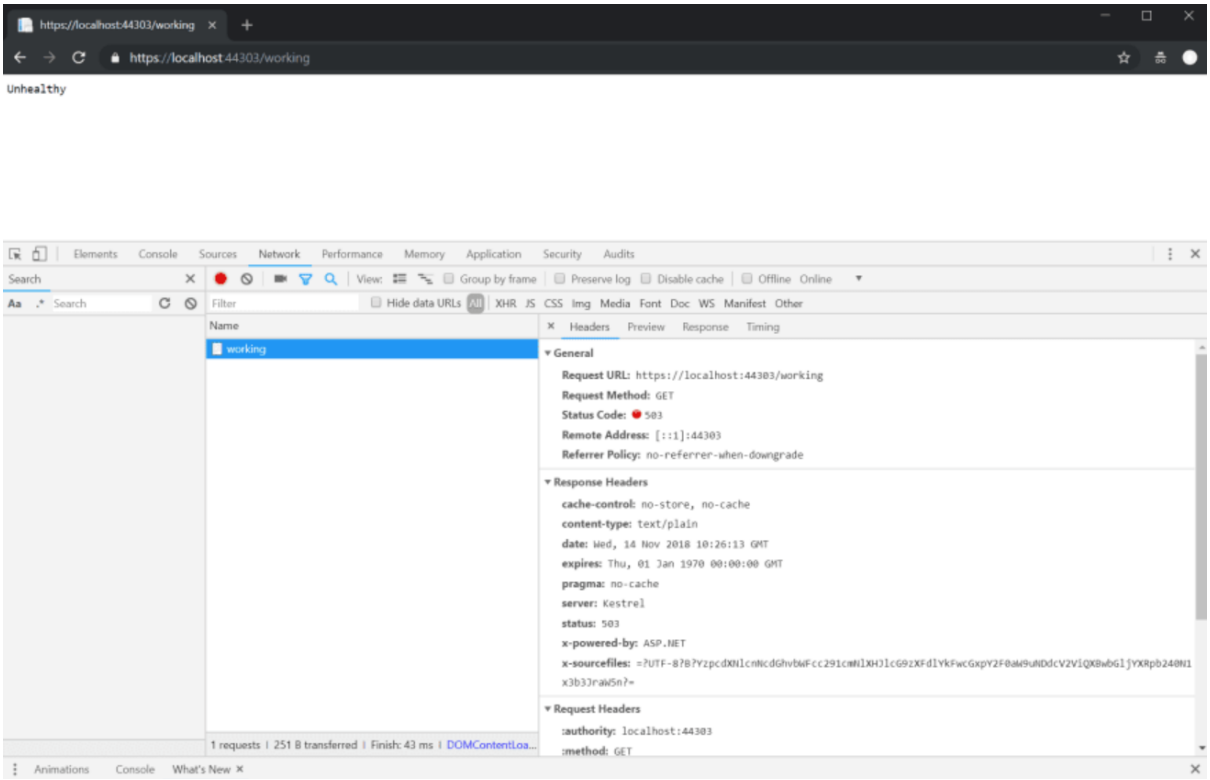
Indicates that the component was healthy the same can be achieved with more understandable response. Healthy message can be customized in below ways,

- Response: General Well-Being
- Component-wise Health Response



■ Unhealthy Message Screen

When a service does not perform as intended, unhealthy status will be returned check. When running the check, an unhandled exception was thrown or application is malfunctioning and offline, unhealthy message will be thrown to end user end. An "unhealthy" check indicates that the component is completely inoperable. For instance, a Redis cache connection could not be made.



6.0Assumptions

When conducting a health check, the end user won't attempt to examine every service, dependency, etc. Considering a health check as essentially a Go/No Go test that informs the load balancer whether the service is online and operational. Load balancers won't be able to recover instances that fail.

7.0Exception Handling

Exception appearing for any failure should be logged into the application logs.