HEALTH CHECKS
IN
SPRING BOOT

TEAM DETAILS

Team Head: Dr. M. Sureshkumar
Associate Professor/I.T.,
Sri Sairam Engineering College

Team Mentor: Ms. V. Narmadha
Assistant Professor/I.T.,
Sri Sairam Engineering College

Team Members:

1. Theerej C
2. Arathi P
3. Sakthivel M
4. Shiva Sankar
5. Vishwa Raviraj S
6. Anurega T R
7. Kishore M A
8. Rajesh N
9. Prasanth K
10. Guru Sanjay R K
11. Sri Ganesh
12. Sunil Raj

College: Sri Sairam Engineering College
Chennai

TECHNICAL SOLUTION APPROACH

CONTENTS

1 INTRODUCTION

The level of innovation and technological growth within the realm of web application development is consistently quickening. It is necessary to do the necessary monitoring of both the Workflow as well as the constant status of each web service, endpoint, and database. In this Section, we will investigate how the widely used Java framework Spring Boot may be modified to facilitate the creation of online and corporate applications such as web services, endpoints, databases, and other components that are associated with these applications. Since Spring Boot needs less setup, the process of building and delivering Spring applications that are suitable for production is simplified by its use. When a programme provides a service that can be used to measure the well-being of a Coherence participant, this may be a useful tool. Further health checks can be implemented by the developer of an application with the use of the API that is offered for this purpose. Monitoring ensures that the planned tasks of a component are carried out in the most effective manner possible. It is important to identify any potential health problems at an early stage and to monitor the API environment, particularly when developing microservices. An application that offers a service that may identify the overall health of a Coherence member may find it helpful to take use of the health check API, which enables application developers to add their own unique health checks.

1.1 ABOUT THIS DOCUMENT

This document includes an overarching perspective of the Health Check API as well as the solution method that is used to achieve the desired level of wellness for the services, endpoints, and any other servers that are relevant to this topic. It provides an explanation of the component diagram, as well as the overall flow diagram and the sequence diagram. Many Test Cases were openly discussed and a detailed solution was developed in accordance with those discussions.

1.1.1. PURPOSE AND SCOPE

The primary purpose of this API development is to monitor how well web services are performing and to spot the problems earlier before they become catastrophic. Having a poor health state will be indicated and helps to restore the operational status of the service, which demonstrates its incapacity to link with other dependent services. This component ensures the independence of services given by this healthcheck API are efficient with a lot more flexibility than the other frameworks. A service instance's health may be frequently checked and health examination of data from APIs may also contain component execution times or times for connecting to downstream services. Use health check APIs to verify the status of your services

and the dependencies on them. Use the health checks to spot failing services and maintain the functionality of the portions of the application that depends on them.

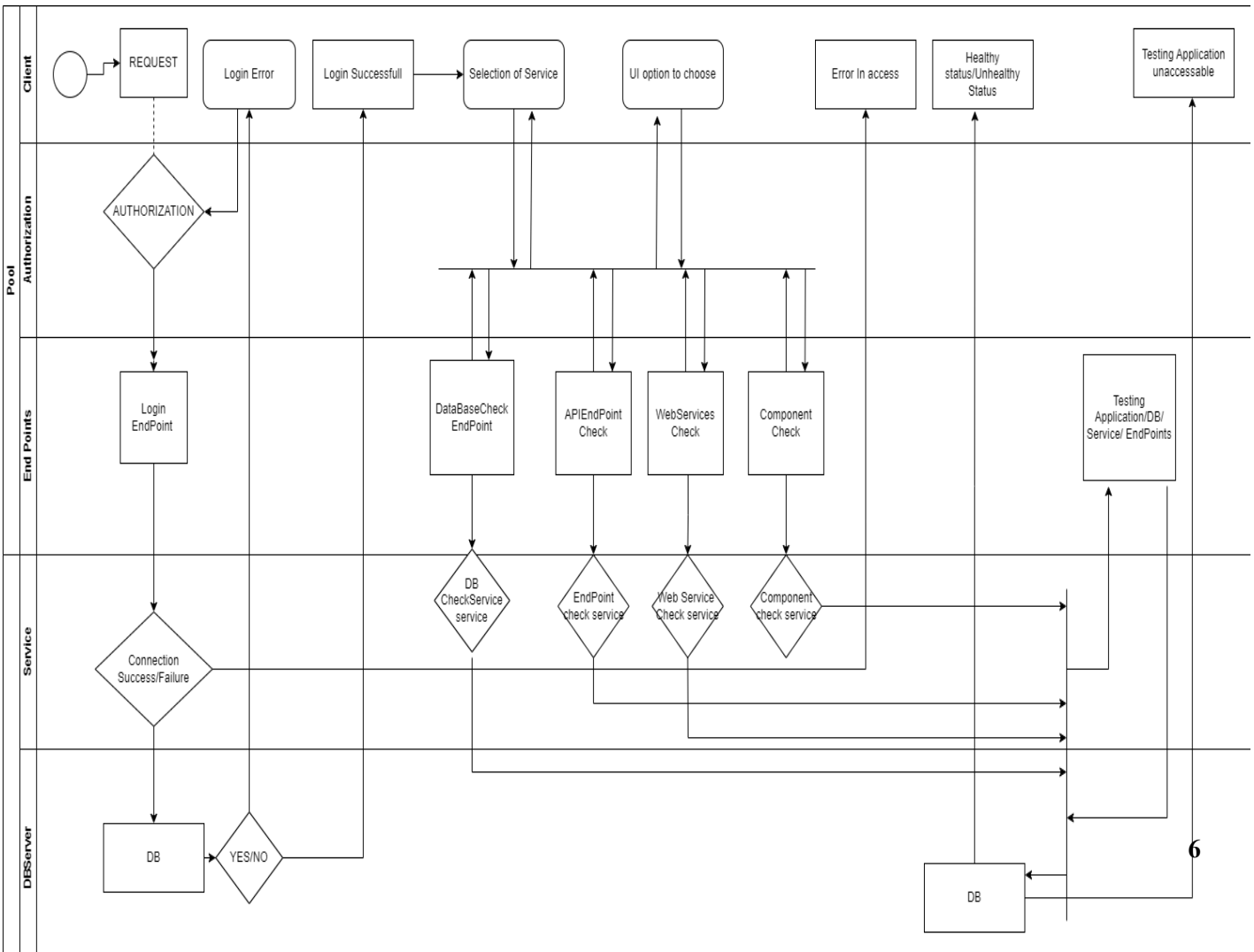
Here are some possible scopes for a health check API using Spring Boot:

Basic health check: This API endpoint can return a simple response indicating the health of the service, such as a HTTP 200 OK response with a "OK" message.
Detailed health check: This API endpoint can return detailed information about the health of the service, such as database connection status, server status, memory usage, etc.
Custom health checks: This API endpoint can include custom checks specific to the application or service being developed, such as checking the availability of a third-party API or service.
Integration with monitoring tools: This API endpoint can be integrated with monitoring tools like Prometheus, Grafana, or Nagios to provide real-time insights into the health of the application.
Security checks: This API endpoint can include security checks to ensure that only authorized users or systems can access the endpoint, such as using OAuth 2.0 or JWT tokens for authentication and authorization.
Logging and error reporting: This API endpoint can be used to log errors and exceptions that occur within the service, as well as to report them to a centralized error tracking system like Splunk or ELK stack.

2. COMPONENT DESIGN

2.1 COMPONENT DESIGN DIAGRAM

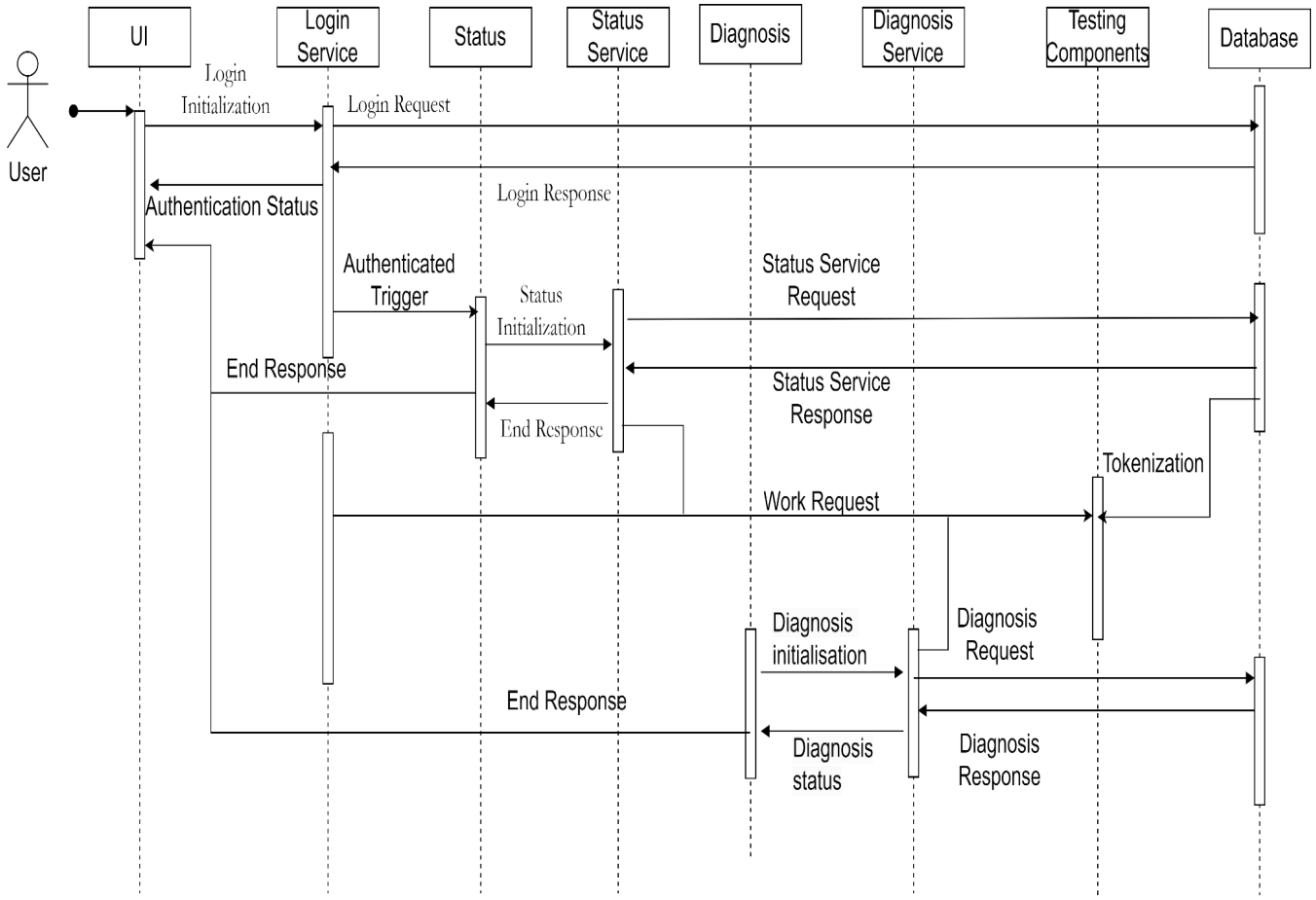
The main part of this component is the end points of this component. The Request of the user comes from the user and it is sent for the authorization in the authorization module. Then the request passes to the login endpoint of the application. This is where the backend of the application begins. Then the request is sent to the DB so that it will be authorized. When the login successfully completes the component can be used to check the parts of other websites. The UI layer will get the request from the user and send the request to the end point specified for the service we want. Then the service layer will check for the application or the component which the user specified. The request will be sent by the service and the request will be received by that component. The request is verified in the DB and the response to the client is then sent to the client. The tested components result is stored in the DB for the future reference. The error is then diagnosed and indicated to the user.



2.1.1 OVERALL WORKFLOW

SEQUENCE DIAGRAM

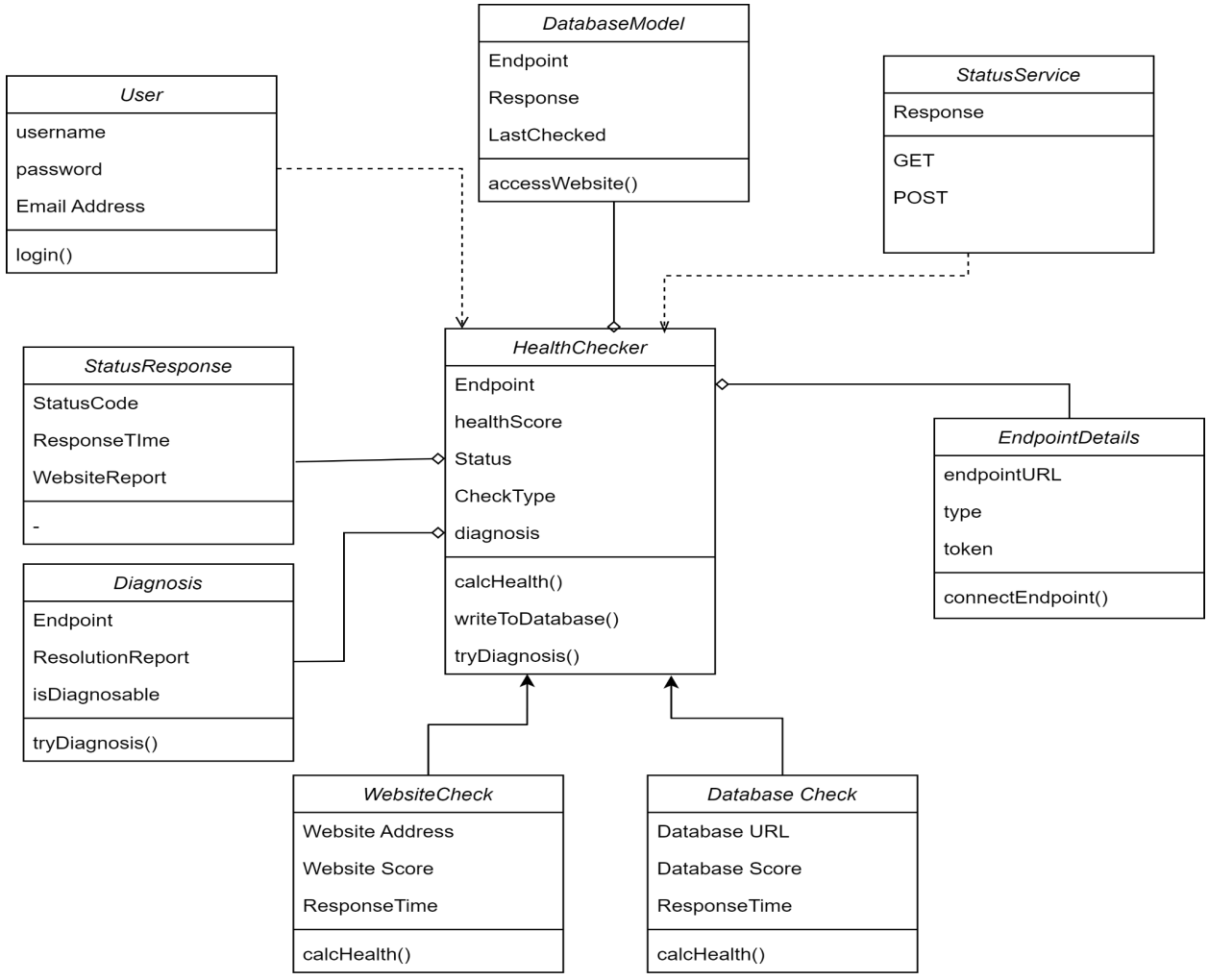
When a user initiates the Login service via the user interface (UI), the action triggers a request to the database to authenticate the user's credentials. The authentication status is fetched from the database and sent to the user interface by the service. At the same time, the authenticated trigger initiates the status service. The database receives a status request through which test components (components, web servers, endpoints, and databases) are tokenized and their status is returned. The Status Service will receive the response. Status in the UI is updated once the end response has been received. When a user indicates an interest in utilizing the Dignosization Option service, a request is sent to the service, and the service responds with additional information. All service modules submit the overall request to all testing components. Each service then provides a final response to the user interface.



2.1.2 LOW LEVEL DESIGN

The primary class of the application is HealthChecker, which is responsible for holding the core logic. The StatusService class is used to provide HTTP request services to the user. StatusService can be used to receive authentication details from the user. The login class is used to make sure that the user provides correct authentication information and grants the user access to the main service. The StatusService Class is also used to accept requests from the client and sends it to the HealthChecker to be processed. The HealthChecker object determines the type of request and processes it accordingly. It accesses the endpoint from the details provided by the users and processes the returned value to determine the health. The information extracted from the endpoint can also be saved in the database. The HealthChecker object can send requests to the endpoint frequently at regular intervals and track the health of the endpoint. Additionally, the diagnosis class can be used to help diagnose the problem, if found in the endpoint. Finally, the details are returned back to the user and saved in the database.

CLASS DIAGRAM



3 TECHNOLOGY & FRAMEWORKS TO BE USED

1. SpringBoot Java

- Spring Boot is a popular framework for building web applications using the Spring framework.
- It is designed to simplify the process of creating stand-alone, production-grade Spring-based applications.
- Spring Boot provides a wide range of features that make it easy to configure and deploy applications quickly and efficiently.
- These features include auto-configuration, which automatically configures many of the components needed for an application to run; starters, which provide pre-configured dependencies for common use cases; and a command-line interface that allows developers to quickly create, run, and test applications.
- Spring Boot is used to build a wide range of applications, including web applications, microservices, and APIs.
- Its ease of use, along with its robust set of features, has made it a popular choice for developers who want to quickly build high-quality, scalable applications.
- In addition, Spring Boot integrates well with other popular technologies such as React, Angular, and Vue, making it a versatile tool for modern web development.

2. React

- React is a JavaScript library used to create user interfaces
- ReactJS is an efficient, flexible, and declarative JavaScript library for building reusable UI components
- It is open-source and focuses only on the view layer of an application
- ReactJS is component-based and uses encapsulated components to manage their own state and create complex UIs
- It allows the passing of rich data through the app while keeping the state out of the DOM since component logic is written in JavaScript rather than templates
- With React, new features can be developed without rewriting existing code
- The primary goal of ReactJS is to enhance app speed by creating user interfaces
- ReactJS uses virtual DOM, a JavaScript object that improves app performance
- ReactJS can be used on both the client and server-side as well as with other frameworks
- It uses components and data patterns that improve the readability and maintainability of larger apps.

3. NoSQL (MongoDB)

- NoSQL (Not Only SQL) is a term used to describe a class of databases that do not rely on the traditional relational database model.
- NoSQL databases were designed to address some of the limitations of relational databases, such as scalability, flexibility, and performance.
- NoSQL databases come in many different flavors, one of which is NoSQL. NoSeQL (Not Only SQL extended by Query Language) is a database technology that combines the best of both NoSQL and SQL worlds.
- NoSQL databases are designed to handle unstructured or semi-structured data that does not fit well in a traditional relational database.
- NoSQL databases allow for a more flexible schema design, as there are no predefined schema requirements.
- This makes it easier to handle data that changes frequently, such as social media data or sensor data.
- NoSQL databases are also designed to handle massive amounts of data, which makes them ideal for big data and real-time applications.
- NoSQL databases use a query language that is similar to SQL, but with some additional features to handle unstructured data.
- These features include support for nested data structures, arrays, and maps.
- In summary, NoSQL is a type of NoSQL database that offers a flexible schema design and can handle massive amounts of unstructured data.
- NoSQL databases use a query language that is similar to SQL, making them accessible to developers with SQL knowledge while also being able to handle unstructured data.
- MongoDB is an open-source NoSQL document database that uses a JSON-like schema instead of traditional table-based relational data. Spring Boot offers several conveniences for working with MongoDB, including the spring-boot-starter-data-mongodb ‘Starter’. Spring Data includes repository support for MongoDB.
- Spring Boot offers auto-configuration for Embedded Mongo. To use it in your Spring Boot application add a dependency on de.flapdoodle.embed:de.flapdoodle.embed.mongo.
- The port that Mongo will listen on can be configured using the spring.data.mongodb.port property. To use a randomly allocated free port use a value of zero. The MongoClient created by MongoAutoConfiguration will be automatically configured to use the randomly allocated port.

4 SOLUTION APPROACH

Health checker is a tool used to monitor the health of an application or system. If the system is healthy it returns the system is in healthy status. If the system is unhealthy it diagnoses the issues that arise when the health check endpoint returns a non-healthy status. The solution approach for creating a health checker using Spring boot:

- To create a User Interface for health checkers using React.js.
- To create a spring application and add the required dependencies.
- Define a health check endpoint using spring boot actuator dependency.Health check endpoint can be configured to return a JSON object with a current status of an application.
- Define the Health check that should be performed by the health check endpoint. The health check can be implemented by a health indicator interface which returns a current health status and you can implement this interface at any custom health check required by an application.
- Configure the health check endpoint to include a custom health check.
- Test the health check endpoint by sending the POST request to the defined endpoints and responses are verified according to the expected information.
- If the response is healthy then the system or application is in Good Condition. If the response is unhealthy then alert pops that a system or application is unhealthy.

Health checker will diagnose any problems that emerge when a health check endpoint returns to a non-healthy state. The Solution approach for diagnosing issues using Spring Boot is:

- To record any failure or problems that occur, configure a logging system.
- If a health check endpoint returns a non-healthy state, note the error message and any other information about the problem.
- Log files are analyzed and issues are identified. This can be done by searching the error message.
- If any major issue has occurred then fix the issue and reboot the application or a system.
- Health check endpoints will be checked again and again till it gets a healthy response from an application or system.

TESTING

1. Database testing

- Checking Database connectivity by verifying whether the database can be accessed or not and by checking if the database server is running or not.
- Databases are a key component of almost every application's back end. A Healthy database will have a stable database connection.
- The health of a database can be checked in many ways. One of the ways is to check to see if transactions and queries are being processed in a timely manner and in a proper manner.
- The testing process is by sending a connection request to Database using our endpoint. Then if the response is good then we return healthy as response. Or we send the result to the diagnosis side.

2. Web service Testing

- Web service connectivity is checked by confirming that the web service is functioning properly and that the service endpoints are reachable or not.
- Check to see if the service responds to valid requests with accurate and anticipated outcomes and make sure to determine if the service handles valid requests properly. verify to determine whether the application properly manages unexpected errors.

- Validating Input Tests:

    - Confirm to ensure that the application generates the proper error messages in response to invalid inputs.
    - Check to see if the site declines requests with incorrect inputs.
    - Verify whether the application validates incoming data properly.

- Verify that service logs are being properly maintained.

3. Testing Components

- Set up health tests for each component you intend to evaluate.
- Examines the component's state and provides a status report showing whether the component is healthy or not.
- Each Component is one of the system's primary parts to ensure that we will send a request to the component. If the component returns a valid response, the component is operating normally otherwise, the component has a problem.
- To check whether the health checker is running properly or not we will first check our health checker by:
- Creating an endpoint. Send a request to the health checker endpoint.

- Examine the response from your health checker to ensure it contains the anticipated status code and response body.
- Examine health checker's logs to ensure that there are no errors or warnings that suggest the health checker is not working Correctly.

4. Testing Endpoints

- Send requests to an application endpoint to implement health monitoring. The application should run the necessary checks and return a status indication.
- The response code is being validated. An HTTP response code of 200 indicates that the application responded without error. To provide more comprehensive results, the monitoring system may also look for other response codes.
- Testing the Endpoint performance by sending multiple requests at the same time and measuring the response time.
- Measuring response time, which reflects a combination of network latency and the time it took the application to process the request. An increasing value may indicate a problem with the application or network.
- Ensure that the API reacts appropriately to error conditions (e.g. 404 not found, 500 server error, etc).