Systems Design

System Monitoring Server

Henry Zektser, Zemenu Guanche, Jorge Castro, Martha Agbugui

Prof. Michael Walsh

11/5/2023

Table of Contents

[Background 4](#_Toc149849401)

[Problem Statement 4](#_Toc149849402)

[Technology Solution 4](#_Toc149849403)

[Process Maps 5](#_Toc149849404)

[Functions and Requirements Table 7](#_Toc149849405)

[User Interface 13](#_Toc149849406)

[Wireframe 1: Status Page 13](#_Toc149849407)

[Description 13](#_Toc149849408)

[Wireframe 2: Server History 14](#_Toc149849409)

[Description 14](#_Toc149849410)

[Wireframe 3: Server Management 15](#_Toc149849411)

[Description 15](#_Toc149849412)

[Wireframe 4: Alert Management 18](#_Toc149849413)

[Description 18](#_Toc149849414)

[Wireframe 5: Authentication 19](#_Toc149849415)

[Description 19](#_Toc149849416)

[Infrastructure Architecture 20](#_Toc149849417)

[Network Topology 20](#_Toc149849418)

[Information Architecture 22](#_Toc149849419)

[Entity Relationship Diagram 22](#_Toc149849420)

[Security and Privacy Architecture 23](#_Toc149849421)

[Programming 25](#_Toc149849422)

[Front-end Technology 25](#_Toc149849423)

[Back-end Technology 25](#_Toc149849424)

[Database and Data Query Language 26](#_Toc149849425)

[Containerization and Development Environment 26](#_Toc149849426)

[Hosting and Cloud Infrastructure 27](#_Toc149849427)

[References 28](#_Toc149849428)

# Background

## Problem Statement

Inefficient system monitoring, disjointed incident management, and inadequate communication during downtime undermine the operational resilience and customer satisfaction (Team, 2022).

## Technology Solution

The proposed solution is a comprehensive systems monitoring server that combines monitoring, alerting, and user visibility in a unified platform. Specifically, the system will allow users to define servers to monitor by providing an IP or an http(s) URL, and a list of users’ email addresses to alert in the event of issues. The system will continually monitor all IPs via ping and URLs via curl, and email alerts in the event of invalid status codes. A status page, showing a list of monitored servers, their current pass/fail state, and any recent issues will also be provided.

## Process Maps

A diagram of a successful audit

Description automatically generated

Figure - Status Page Process Map

A diagram of a successful author

Description automatically generated

Figure - Server History Process Map

A diagram of a software process

Description automatically generated

Figure - Server Management Process Map

A diagram of a server

Description automatically generated

Figure - Alert Management Process Map

A diagram of a flowchart

Description automatically generated

Figure - Server Monitoring Process Map

A diagram of a process

Description automatically generated

Figure - Authentication Process Map

## Functions and Requirements Table

|  |  |
| --- | --- |
| **FUNCTION** | **REQUIREMENTS** |
| **Status Page** | * The system will verify if the status page is configured to be public, and enforce authentication prior to display if it is not. * The system will display a list of all monitored servers. * The system will display a current status (up/down) for all monitored servers. * The system will use iconography and color coding to display current status – green for up, and red for down. * The system will display the last check time for each server. * The system will display the time from last check for each server in seconds. * Where the time from last check is greater than 60 seconds, the system will round the displayed time to minutes (if less than 60 minutes), hours (if less than 24 hours), or days (if greater than 24 hours). * The system will provide a link from each server to it’s history page. * The system will provide a link to add a new server. * For authenticated users with administrator permissions, the system will provide a link to manage users. * For authenticated users with administrator permissions, the system will provide a link to manage alerts. |
| **Server History** | * The system will enforce authentication prior to display. * The system will display known server information (hostname, IP address, owner). * The system will display a timeline chart of prior incidents for the last 30 days. * The system will display all prior incidents in descending date order – newer incidents on top, older ones below. * For each historical incident, the system will display any incident notes entered into the system. * For each historical incident, the system will display the start and end time of the incident. * For each historical incident, the system will compute and display the length of the incident in minutes. * For each historical incident where the length of incident exceeds 60 minutes, the system will round the displayed time to hours (if less than 24 hours), or days (if greater than 24 hours).   + For each historical incident the system will display any alerts that were sent, and whom they were sent to. |
| **Server Management** | * The system will enforce authentication prior to display. * The system will allow entry of new servers. * The system will allow editing of existing servers. * The system will allow deletion of existing servers. * The system will verify administrator permissions prior to allowing add/edit/delete actions. * The system will prompt for server name, optional description, and hostname or IP address. * The system will prompt for the time interval at which the check should be performed. * The system will prompt for the type of check to be performed. * For HTTP checks, the system will prompt for the URL. * For HTTPS checks, the system will prompt for the URL and whether SSL certificate validity should be enforced. * For HTTP and HTTPS checks, the system will prompt for the expected status code. * For TCP checks the system will prompt for a port number. * The system will log all server creation, editing, and deleting actions to the audit log, including time, user performing the action, and action performed. |
| **Alert Management** | * The system will enforce authentication prior to display. * The system will allow entry of new alert recipients. * The system will allow editing of existing alert recipients. * The system will allow deletion of existing alert recipients. * The system will verify administrator permissions prior to allowing add/edit/delete actions. * The system will prompt for recipient name and email address. * The system will allow selection of specific servers or “all servers” to alert for. |
| **Server Monitoring** | * The system will iterate through all configured servers continuously. * The system will store the time it runs every check. * The system will execute a check when the current time subtracted from the last check time exceeds the check interval. * For HTTP checks the system will attempt to fetch the URL and compare the status code to the expected code. * For HTTPS checks the system will attempt to fetch the URL and compare the status code to the expected code, ignoring or verifying certificate validity per check configuration. * For HTTP and HTTPS checks, the system will register success when the response code matches the expected code. * For HTTP and HTTPS checks, the system will register a failure and issue an alert when the response code does not match the expected code, or connectivity fails. * For TCP checks the system will attempt to open a TCP connection to the configured port. * For TCP checks, the system will register success when a connection can be opened. * For TCP checks, the system will register a failure and issue an alert when a connection fails to be opened. * For Ping checks, the system will send an ICMP echo request to the configured hostname or IP address. * For Ping checks, the system will register success when an ICMP echo response is received. * For Ping checks, the system will register a failure and issue an alert when an ICMP echo response is not received or contains an error code. |
| **Authentication** | * The system will require unauthenticated users to authenticate prior to access. * The system will redirect users to the organizations IdP for password/2FA verification. * The system will display a login failure message when the IdP fails to authenticate. * The system will allow access and display the requested page when the IdP successfully authenticates. * The system will provide administrator permissions to those users who are configured to have them. * The system will disallow administrator actions to those users who do not have sufficient permissions. |

# User Interface

## Wireframe 1: Status Page

### Description

The status page function is designed to enable the user to see a birds-eye view of all monitored servers within the environment (*What Is a Statuspage and Why Do You Need It | Freshstatus*, n.d.). The page will display all servers, their status (up or down), and the length of time of the last check for that server. This page also displays an inventory (by count) of current Servers, Alerts, and Users registered within the system. Finally, this page displays a summary of recent events within the system, including successful and failed checks within the past hour. This page will also serve as the home page for the system and the entry point for creating additional system monitors and viewing the history of a given server.

A screenshot of a computer

Description automatically generated

Figure - Status Page Wireframe

## Wireframe 2: Server History

### Description

The server history page is designed to enable the user to see all outage events for a given server for the past 30 days. This page will show past events along with any associated event notes, providing context for prior issues that may be related to a current outage. This page will also display current status information for the server, similar to the status page. Finally, this page will provide a visual timeline of past outages, via graph, to allow users to quickly visualize if issues may be occurring at a specific interval, e.g., every Friday night.   
A screenshot of a computer

Description automatically generated

Figure - Server History Wireframe

## Wireframe 3: Server Management

### Description

The server management page is designed to enable administrators to quickly add new servers to the system for monitoring. This page will also enable editing of existing servers already being monitored. This page also serves as the method to define the type of check to be run on the server, and how often the check should be performed. HTTP or HTTPS checks will run a basic HTTP GET operation, checking for an expected status codes. TCP checks will ensure a TCP connection can be established to a specific port. Ping checks will perform a simple ICMP echo request and check for an appropriate ICMP response.

A screenshot of a computer

Description automatically generated

Figure - Server Management Wireframe (Base)

A screenshot of a computer

Description automatically generated

Figure - Server Management Wireframe (HTTP Setup)

A screenshot of a computer

Description automatically generated

Figure - Server Management Wireframe (TCP Setup)

A screenshot of a computer

Description automatically generated

Figure - Server Management Wireframe (Ping Setup)

Screens screenshot of a phone

Description automatically generated

Figure - Server Management Wireframe (Delete w/Confirmation)

## Wireframe 4: Alert Management

### Description

The alert management page is designed to enable administrators to quickly add new alert recipients to the system for monitoring. This function will also enable editing of existing recipients, as well as removing recipients. In addition to specifying the recipient via email address, this page also enables administrators to select either all servers, or a subset of servers for which to send alerts. This enables targeted alerting to reach those who need to be notified without bombarding every recipient with every alert.

A screenshot of a computer

Description automatically generated

Figure - Alert Management Wireframe (Base)

Screens screenshots of a computer

Description automatically generated

Figure - Alert Management Wireframe (Delete w/Confirmation)

## Wireframe 5: Authentication

### Description

The authentication page is designed to ensure that only authorized users can access the system. This login page relies on an external identity provider (IdP) integration, allowing organizations to leverage their existing single sign-on (SSO) solution for password verification, and optionally two-factor (2FA) authentication.

A screenshot of a login screen

Description automatically generated

Figure - Authentication Wireframe

# Infrastructure Architecture

## Network Topology

A computer server diagram with a computer

Description automatically generated with medium confidence

Figure - Network Topology

The proposed technology solution is a robust and highly available system designed to ensure the security, reliability, and scalability of a web-based application. The infrastructure is composed of several key components, including a Firewall/Web Application Firewall (WAF), redundant UI servers, redundant application servers, and a clustered MongoDB database, all supported by load balancers at each layer. This architecture is engineered to meet the demanding requirements of modern web applications.

1. **Firewall/WAF Layer**: At the forefront of the infrastructure is a Firewall/WAF combination. This layer serves as the first line of defense against cyber threats and malicious traffic. It provides essential security measures such as intrusion detection and prevention, traffic filtering, and DDoS protection. The WAF component specifically safeguards web applications from common vulnerabilities and exploits. By actively monitoring incoming and outgoing traffic, the Firewall/WAF ensures data integrity, confidentiality, and system availability.
2. **UI and Application Server Layers**: The UI and application server layers consist of two sets of redundant servers. Redundancy is vital for maintaining high availability. These servers host the user interface and the core application logic. Load balancers distribute incoming traffic evenly across the servers, ensuring optimal resource utilization and minimizing downtime. In the event of a server failure, requests are seamlessly rerouted to the healthy servers, minimizing service interruptions.
3. **MongoDB Database Layer**: The database layer features a cluster of three MongoDB servers. Clustering enhances data reliability, scalability, and fault tolerance. Load balancers are strategically placed in front of the MongoDB servers to evenly distribute read and write requests. This not only optimizes database performance but also ensures data consistency. By maintaining multiple copies of data across different nodes, the clustered MongoDB system provides resilience against hardware failures.
4. **Load Balancers**: Load balancers are positioned at every layer of the infrastructure. They play a pivotal role in optimizing resource utilization and improving system performance. Each load balancer ensures even distribution of incoming requests, prevents server overload, and redirects traffic to healthy servers in case of a failure. This approach helps in load distribution and provides an additional layer of fault tolerance and high availability.

# Information Architecture

## Entity Relationship Diagram

A screenshot of a computer

Description automatically generated

Figure - ER Diagram

The information architecture for this project is simple. Users and Servers are stored as primary entities. A user is bound to servers for alerting purposes via the UserAlert entity. The three supported check types (HTTP, TCP, and Ping) are stored as discrete entities that are keyed to a server. It should be noted that the checks are created with a one-to-many relationship to a server. While not in-scope for this development effort, the schema is set up deliberately to enable a future enhancement wherein a server can be checked in multiple ways. For instance, a ping check to ensure the server is up, and a TCP check to then further ensure an application on that server has not crashed.

# Security and Privacy Architecture

A number of security risks are envisioned in this solution, and compensating controls have been designed to address them as follows:

**Security Risk 1: Unauthorized Access (Confidentiality)**

**Control: Access Control --** To mitigate the risk of unauthorized access and protect the confidentiality of sensitive data, the solution will implement robust access control mechanisms. These controls will include user authentication and authorization systems. Users will be required to log in with unique credentials, and role-based access control (RBAC) will ensure that they can only access the resources and data necessary for their roles. This will prevent unauthorized users from gaining access to sensitive information and maintaining confidentiality.

**Security Risk 2: Data Tampering (Integrity)**

**Control: Data Validation, Integrity Checks, and Auditing --** To address the risk of data tampering and ensure data integrity, the technology solution will employ data validation and integrity checks at multiple levels. This control involves validating input data on the UI servers, implementing secure data transfer protocols (such as HTTPS) to protect data in transit, and using cryptographic hashing to verify data integrity when it's stored in the MongoDB database. Regular integrity checks will be performed to detect any unauthorized alterations or corruption of data, maintaining its integrity (Taylor, 2023). All actions within the system will be logged to further establish an audit trail for all data changes.

**Security Risk 3: Distributed Denial of Service (DDoS) Attacks (Availability)**

**Control: DDoS Mitigation and Redundancy --** To safeguard against DDoS attacks and maintain system availability, the technology solution will employ a multi-faceted approach. The Firewall/WAF layer will be configured to detect and mitigate DDoS attacks in real-time (Official, 2022). Additionally, the use of load balancers at each layer provides distribution and redirection of traffic to healthy servers, which can help absorb traffic spikes. The infrastructure also includes redundant UI and application servers to ensure that even if one server is under attack, the system can continue to function. Cloud-based DDoS protection services may also be considered for an extra layer of protection during sudden traffic surges.

By implementing these controls, the solution addresses the key aspects of information security: Confidentiality, Integrity, and Availability. These controls work together to provide a comprehensive security framework that safeguards sensitive data, ensures data remains unaltered, and guarantees that the system remains accessible to authorized users even in the face of security threats.

# Programming

This project leverages a number of open-source technologies across the tech stack in order to accelerate development timelines:

## Programming Languages

1. **TypeScript**: TypeScript was chosen for its ability to add strong typing to JavaScript, catching errors at compile-time rather than runtime. This reduces the likelihood of bugs, enhances code quality, and facilitates collaboration among developers (*JavaScript With Syntax for Types.*, n.d.).
2. **Golang (Go)**: Golang was chosen for its efficiency, strong performance, and simplicity. Go's statically typed nature and built-in concurrency support make it an excellent choice for building scalable and performant back-end API services. Go has strong support for GraphQL and JSON data, allowing easy interchange of data with the other chosen technologies. It also has a vibrant open-source community and a focus on writing clean and maintainable code (Barney & Gillis, 2023).

## Front-end Technology

1. **React**: React is a popular and well-supported library for building user interfaces. Its component-based architecture promotes reusability and modularity, allowing for efficient development and maintenance of complex UIs. Additionally, React's virtual DOM improves performance by minimizing unnecessary re-rendering.
2. **Patternfly:** Patternfly is a UI framework that provides out-of-the-box components for enterprise application interfaces. It’s tight integration with TypeScript and React enables engineers to quickly assemble attractive UIs without significant investment in basic web development (*PatternFly • About PatternFly*, n.d.).

## Back-end Technology

1. **Node.js**: Node.js is a server-side JavaScript runtime that offers non-blocking I/O and a single-threaded event loop, making it highly efficient for handling concurrent connections. Its ecosystem is rich in libraries and frameworks, including Express.js.
2. **Express.js**: Express.js is a minimal and flexible Node.js web application framework that simplifies building robust, RESTful APIs. Its middleware architecture streamlines the development process and supports various extensions and plugins.

## Database and Data Query Language

1. **MongoDB**: MongoDB is a NoSQL document database that provides flexibility and scalability. Its schema-less design is well-suited for projects with evolving data structures, and it offers horizontal scaling options (MongoDB, n.d.). Given the flexibility of GraphQL, which can accommodate changes in the data schema, MongoDB complements the technology stack nicely.
2. **GraphQL**: GraphQL was chosen as the query language for its flexibility and efficiency in data retrieval. It allows clients to request exactly the data they need, reducing over-fetching and under-fetching of data. This is particularly beneficial in front-end development, where minimizing data transfer enhances application performance (*Introduction to GraphQL | GraphQL*, n.d.).

## Containerization and Development Environment

1. **Docker**: Docker containers provide a consistent and reproducible environment across different stages of development, from local development to production deployment. Containerization simplifies the setup of the development and production environments, ensuring that the application behaves consistently in various settings.
2. **JetBrains GoLand**: GoLand is an IDE specifically designed for Go development. It offers a rich set of features, including code navigation, intelligent code completion, and debugging tools tailored to the Go language (*GoLand by JetBrains: More Than Just a Go IDE*, 2021). Using an IDE like GoLand improves developer productivity and code quality.

## Hosting and Cloud Infrastructure

1. **Amazon Web Services (AWS)**: AWS is a leading cloud service provider known for its reliability, scalability, and extensive suite of services. AWS Elastic Beanstalk, for instance, simplifies deploying and managing containerized applications, making it an ideal choice for hosting our Docker containers. AWS also provides infrastructure as code (IaC) tools like AWS CloudFormation for defining and deploying resources in a repeatable and automated manner.

# References

Barney, N., & Gillis, A. S. (2023). Go programming language. *IT Operations*. https://www.techtarget.com/searchitoperations/definition/Go-programming-language

*GoLand by JetBrains: More than just a Go IDE*. (2021, June 10). JetBrains. https://www.jetbrains.com/go/

*Introduction to GraphQL | GraphQL*. (n.d.). https://graphql.org/learn/

*JavaScript With Syntax For Types.* (n.d.). https://www.typescriptlang.org/

MongoDB. (n.d.). *What is MongoDB?* https://www.mongodb.com/what-is-mongodb

Official, A. (2022). How do I mitigate DDoS attacks using AWS WAF? *Amazon Web Services, Inc.* https://repost.aws/knowledge-center/waf-mitigate-ddos-attacks

*PatternFly • About PatternFly*. (n.d.). PatternFly Docs. https://www.patternfly.org/get-started/about/

Taylor, S. (2023). Data validation. *Corporate Finance Institute*. https://corporatefinanceinstitute.com/resources/data-science/data-validation/

Team, P. F. (2022). Importance of having a good monitoring system. *Pandora FMS Monitoring Blog*. https://pandorafms.com/blog/why-you-need-a-monitoring-system/

*What is a statuspage and Why do you need it | Freshstatus*. (n.d.). https://www.freshworks.com/status-page/why-status-page/