PROJECT ABSTRACT

# Website Uptime Monitor with Alert Dashboard

*A web-based uptime monitoring service: multi-region checks for websites, APIs and ports, response-time charts, incident logs, and sub-60-second alerts via Slack, SMS and email, plus public status pages. Delivered built-to-order with the full web application, check engine, report, PPT and viva kit.*

## 01. Title

Website Uptime Monitor with Alert Dashboard.

## 02. Introduction / Background

Downtime is revenue loss and reputation damage, yet many small businesses discover outages from angry customers, not from monitoring. Professional uptime monitoring checks targets from multiple regions on a schedule, distinguishes real outages from single-network blips, and alerts the right channel within a minute. This project builds that service: configurable monitors, a distributed-style check engine, alerting with escalation, and public status pages.

## 03. Problem Statement

Without monitoring, teams learn about downtime late and debug blind — no record of when it started, which regions were affected, or whether it was DNS, TLS or the app server. Single-location pings cause false alarms. The project delivers a complete monitor: HTTPS/API/port/keyword checks from multiple regions, confirmation logic requiring agreement before alerting, response-time history with outage markers, incident timelines with root-cause notes, and status pages that keep customers informed during incidents.

## 04. Objectives

- Support monitor types: HTTPS, API keyword, TCP port, ping, and SSL-expiry checks.
- Implement scheduled checks from multiple regions with quorum-based alerting.
- Deliver alerts within 60 seconds via Slack, SMS, email and webhooks.
- Record response-time series and render outage-marked charts.
- Maintain incident logs and public/private status pages.

## 05. Proposed Methodology

The service has three parts. (1) Check engine: a scheduler dispatches checks per monitor interval; workers in multiple regions perform the probe (HTTP status, keyword presence, TLS validity, port open) and report results with timing. (2) Decision logic: a target is marked down only when a quorum of regions agrees across consecutive checks, suppressing false positives; flapping detection avoids alert storms. (3) Notification and presentation: alert dispatcher fans out to configured channels with escalation; the dashboard shows per-monitor cards, response-time charts, and an incident log; status pages aggregate monitor states publicly.

## 06. System Architecture / Working

Each monitor defines target, type, interval and alert rules. The scheduler queues check jobs; regional workers execute probes and store (timestamp, region, up/down, latency, detail). The

evaluator applies quorum rules — e.g. 2 of 3 regions failing twice in a row — then opens an incident and fires alerts (Slack + SMS + email) with the failure detail. Recovery requires consecutive successes before the incident auto-closes, and every state change is logged. The detail view plots 24-hour response times with the outage gap visible, and the incident table records cause and resolution for postmortems.

## 07. Software Requirements

- Backend: Python (FastAPI/Django) or Node.js; task queue (Celery/RQ) for scheduling
- Database: PostgreSQL/MySQL or time-series store for check results
- Check workers: lightweight probers deployable per region (simulated multi-region in the academic build)
- Frontend: HTML5, CSS3, JavaScript dashboards with charts
- Alert integrations: Slack webhook, SMS gateway, SMTP

## 08. Expected Outcomes

A working monitoring service: monitors with live status, quorum-based alerting demonstrated against a test target, response-time charts with outage markers, incident history and a status page. The quorum/false-positive design is the key viva discussion.

## 09. Applications

- Small businesses and startups without dedicated SRE
- Agencies monitoring client sites under maintenance contracts
- Students learning distributed-systems monitoring concepts

## 10. Future Scope

- Real multi-region deployment on cloud VMs
- Synthetic transaction checks (login flows, checkout flows)
- On-call rotation and escalation policies