

# Assignment #1

*Big Data and Cloud Computing*

Bárbara Neto, up202106176

João Varelas, up201609013

Leandro Costa, up202408816

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>System Design</b>	<b>1</b>
2.1	Architecture Overview . . . . .	1
2.2	Component Interactions . . . . .	2
<b>3</b>	<b>Implementation</b>	<b>2</b>
<b>4</b>	<b>Evaluation</b>	<b>3</b>
4.1	Load Testing . . . . .	3
<b>5</b>	<b>Conclusions</b>	<b>3</b>

## 1 Introduction

The goal of this project is to explore how cloud computing services can be utilized to develop scalable, fast, and highly available backend systems. Specifically, we design and implement a

backend service for a hospital management system, inspired by the MIMIC-III database. The system supports operations for managing patient data, medical admissions, progress reports, media files, and doctor-patient communication through questions and replies. All backend functionality adheres to standard REST API conventions.

To ensure robustness and scalability, the system leverages Google Cloud Platform (GCP) services. AppEngine is used to deploy RESTful services, Cloud Storage for managing media (BLOB) files, and a database service for structured data storage. Additionally, Google Cloud Functions are employed for serverless computing tasks, such as periodic updates and maintenance functions.

The backend supports essential operations such as creating and updating patients, admissions, and medical progress, while also managing media uploads/downloads, patient questions and calculating waiting times.

---

## 2 System Design

### 2.1 Architecture Overview

The system architecture is designed around RESTful principles and leverages multiple Google Cloud Platform (GCP) services to achieve scalability, reliability, and high availability. The main components and their roles are outlined below:

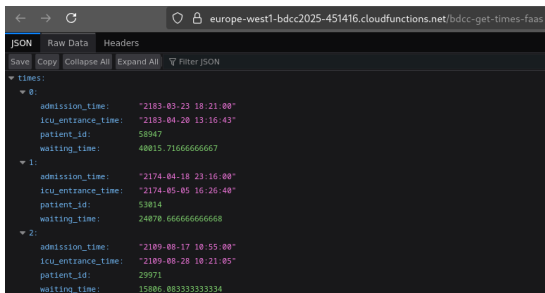
- **Google App Engine (GAE):** Hosts the REST API, implemented in Python 3 using the Flask web framework. It provides CRUD endpoints for managing patients, admissions, progress notes, media, and questions. GAE's auto-scaling capability ensures the system adapts to varying

loads efficiently.

- **Cloud SQL (MySQL):** A fully managed relational database used for storing structured data, including patients, admissions, progress, and questions. The schema is a simplified adaptation of MIMIC-III, consisting of tables such as `PATIENTS`, `ADMISSIONS`, `PROGRESS`, and `QUESTIONS`. Secure internal communication between GAE and Cloud SQL is achieved via Private IP configuration and the VPC Serverless Access Connector.
- **Cloud Storage Buckets:** Used for storing

unstructured data such as images, videos, and documents associated with patients. The REST API provides endpoints for uploading and retrieving files, interfacing with Cloud Storage using GCP APIs.

- **Google Cloud Functions (FaaS):** Intended for periodic or compute-intensive tasks, such as updating a list of patients with the longest waiting times, which is derived from admission data. These functions are triggered via HTTP requests and interact with Cloud SQL in a serverless manner.



```

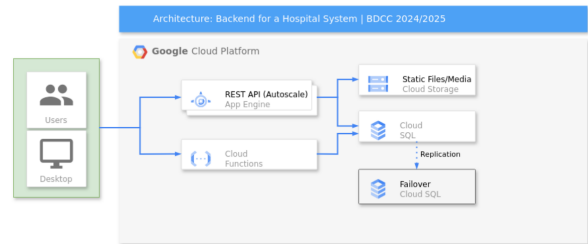
{
  "times": [
    {
      "admission_time": "2183-03-23 10:21:00",
      "icu_entrance_time": "2183-04-20 13:16:43",
      "patient_id": 58947,
      "waiting_time": 48015.71666666667
    },
    {
      "admission_time": "2174-04-18 23:16:00",
      "icu_entrance_time": "2174-05-05 16:26:40",
      "patient_id": 53014,
      "waiting_time": 24078.666666666668
    },
    {
      "admission_time": "2109-08-17 10:55:00",
      "icu_entrance_time": "2109-08-28 10:21:05",
      "patient_id": 29971,
      "waiting_time": 15806.083333333334
    }
  ]
}

```

## 2.2 Component Interactions

To ensure seamless data flow and scalability, the system components are tightly integrated through a modular architecture. At the core,

the REST API hosted on Google App Engine (GAE) is structured into route-specific Blueprints—`patient`, `admission`, `progress`, `media`, `question`, and `waiting time`. Each Blueprint encapsulates logic for handling specific entities, promoting modularity, maintainability, and clarity in code organization.



A key feature of the system is its dynamic database connectivity, determined by the execution environment. In production, the system securely connects to Cloud SQL using the Cloud SQL Python Connector over a Private IP via the VPC Serverless Access Connector, with `pymysql` as the database driver. This connection logic is encapsulated in a reusable function that retrieves configuration from environment variables to initialize secure communication with the database.

## 3 Implementation

Complementing the structured data stored in Cloud SQL, unstructured data such as images, videos, and documents linked to patients is managed via Google Cloud Storage. A dedicated bucket, specified through environment variables, serves as the repository for this media. The `media` Blueprint exposes endpoints for file operations, utilizing GCP client libraries for seamless integration. Upon upload, each file is assigned a universally unique identifier (UUID), ensuring distinct naming in Cloud Storage, and corresponding metadata is stored in Cloud SQL for reference and traceability.

Across all routes, the system follows a standardized pattern for interacting with Cloud SQL. Secure connections are established based on the environment, SQL queries—often involving

joins—are executed, and the results are processed in Python (e.g., calculating waiting times or formatting timestamps) before being returned as JSON responses. Connections and cursors are always closed properly to maintain efficiency. This robust pattern is also leveraged by Cloud Functions for asynchronous tasks, ensuring consistent, secure, and scalable data access while abstracting the complexity of storage operations from clients.

In the implementation phase, we leveraged Google Cloud Functions to deploy a Function-as-a-Service (FaaS) solution, enabling us to handle HTTP requests to query data from our Cloud SQL instance in a scalable, serverless manner. We created a function that establishes a private IP connection to the database hosted on Google Cloud, ensuring data remains within our VPC.

This connection is used by the deployed function *get times*, which is invoked via HTTP trigger. The deployment process uses the *gcloud functions deploy* command with environment variables passed in for secure access, allowing us to manage database credentials and configuration dynamically. By using FaaS, we minimized infrastructure overhead while enabling responsive and cost-effective access to our backend database.

Additionally, the raw dataset—originally in CSV format—was adapted and ingested into Cloud SQL through custom ETL scripts. These scripts

handle data validation, type conversion, and batching, ensuring efficient and error-resilient loading of large datasets like patients, admissions, lab events, ICU stays, etc. Depending on the environment flag *IS PROD*, connections are dynamically routed to either local or cloud databases. Throughout the process, we ensured data consistency by parsing datetime fields, managing nulls, and committing transactions in batches for optimal performance. This dual setup supports both local development and production deployment seamlessly.

---

## 4 Evaluation

### 4.1 Load Testing

The system's performance was evaluated using Artillery, a robust load-testing tool, to assess its scalability and responsiveness under heavy traffic. The tests simulated multiple concurrent users performing actions such as creating patient records, uploading media, and querying data. Key performance indicators, including response times, throughput, and error rates, were closely monitored.

To ensure scalability, Google App Engine (GAE) was configured to automatically scale instances

based on CPU usage and request volume. Cloud SQL was set up with read replicas and automatic storage expansion to handle increasing data loads.

High availability was tested by deploying the system specifically on the europe-west1 region. Latency and failover behavior were measured to ensure consistent performance during potential regional disruptions.

The system is expected to handle hundreds of requests per second, with sub-second response times under normal conditions.

---

## 5 Conclusions

This project demonstrates the effective use of GCP services to build a scalable and reliable hospital backend system. Google App Engine provides a flexible environment for the REST API, while Cloud SQL ensures secure data management with high availability. Cloud Storage efficiently handles multimedia files, and Google Cloud Functions optimize real-time data processing.

By leveraging these tools, the system can manage hospital workflows like patient admissions, medical records, and progress tracking, ensuring performance under high demand through load testing and multi-region deployment.

Overall, the system is designed for scalability, reliability, and some security.