UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Benchmarking Service Architectural Design Specification

ProCoders

Bongani Tshela - 14134790

Harris Leshaba - 15312144

Joseph Letsoalo - 15043844

Minal Pramlall - 13288157

Contents

1 Introduction

1.1 Purpose

This document is intended to provide clarity to the choices made in terms of architectural design for the benchmarking project. Intended users for this document would be any stakeholders of the system, from a developer to the end-user seeking to understand the architecture.

1.2 Document Convention

The Benchmarking uses Java as the programming language of choice to build the system and as such uses Javadoc documentation for its source code.

1.3 Project Scope

The system provides a web interface for users to request and specify the benchmarking services they need. The services will be provided by executing the requested benchmarks on isolated machines where the side-effects that are not a concern of the specified benchmark is minimized. The system measures a variety of performance attributes such as CPU time, elapsed time, memory usage, power consumption, heat generation, etc. It accepts source code for a variety modern programming languages. Limitations for the system are the number of programming languages that are supported and the representation formats the results are represented in.

1.4 References

2 System description

Benchmarking is a very common and useful service, and yet few benchmarking tools or services are easily available. Those that are available may require intricate configuration that is beyond the reach of the budding developers, researchers, teachers and students who would like to use them. The development of a benchmarking service which can be used in a simple and generic way would therefore be welcomed by a large potential user base.

3 Overall Architecture

3.1 Architectural Patterns

The Benchmarking service will utilize a combination of architectural patterns. The first being Client-server architecture, since the clients will interact with a web interface to the server and send through requests for operations to be performed, the server does not know the client and the clients do not know each other. The second pattern is that of a Cloud computing architecture, where it will operate with the Benchmarking program performing as software as a service (SaaS). Resources allocated will be approximately the amount needed, so if the user is charged for usage, then it would be for the actual amount of resources used.

3.2 Architectural Strategies

3.3 Reference Architectures

Reference Architectures for the system would be Java SE and PHP Library architectures.

3.4 Technology and frameworks selection

4 Details of Subsystem

5 Policies

6 Traceability Matrix

7 Project Scope

The core of the system is a benchmarking tool which. The high level modules and their responsibilities are shown in Figure 1

8 User management module

8.1 Scope

The scope of the user management module is shown in Figure 1. The user management module is responsible for maintaining information about the registered users of the system, including the authority levels of each user. User can manage their benchmarking records like deleting and storing new records, view them and retrieving old algorithm they have stored.

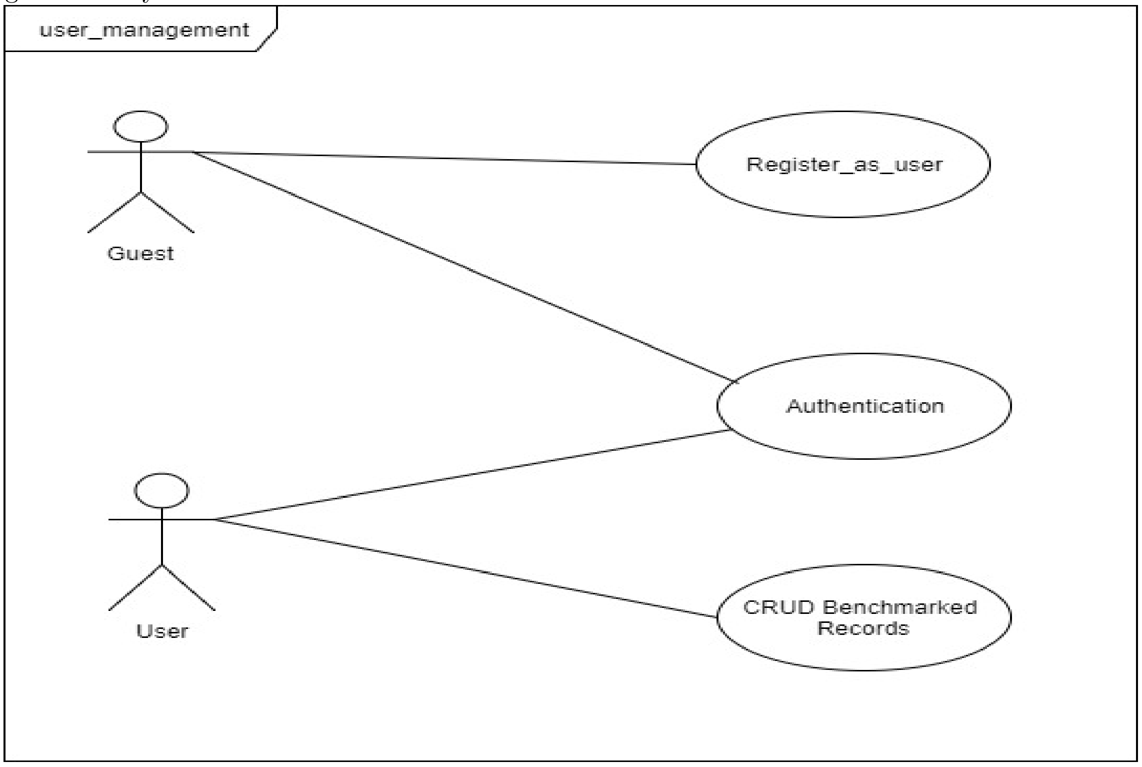


Figure 1 use case.

No information is stored for the guest user. When accessing the service, the user assumes the guest role without being logged in. The guest user may use public services and my register or log in. When a user is registered, the fields shown in the domain model are stored for the user using a unique automatically assigned ID. The user provides all other fields. The password is stored in encrypted format. The user can access all the services provided by the service The register as user use case is initiated by an end-user to create an account.

A user will not be created if the email specified in the request to create a user is already associated with an existing user. This is done to ensure that if a user is already associated with the email address, the user is given access to his/her profile instead of creating a new user.

Note that this use case will use the **getUser(userEmail)** service provided by the module to confirm if another user with that same email already exists in the system. A user should only be created if **getUser(userEmail)** throws the *noSuchUser* exception. After the account has been created a notification will be sent to the user. The system have two types of users: Guest user and Registered user.

8.1.1 Domain model

The users management domain model is shown in Figure 2

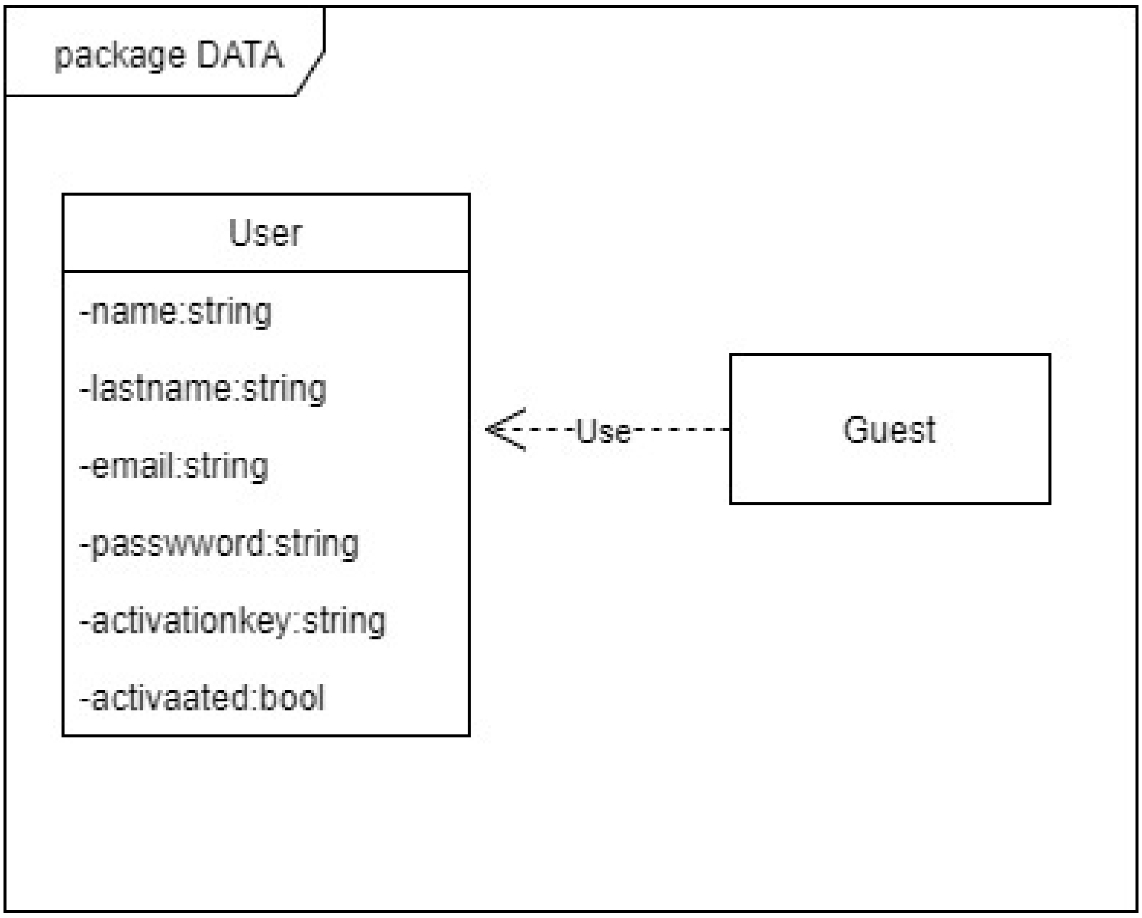
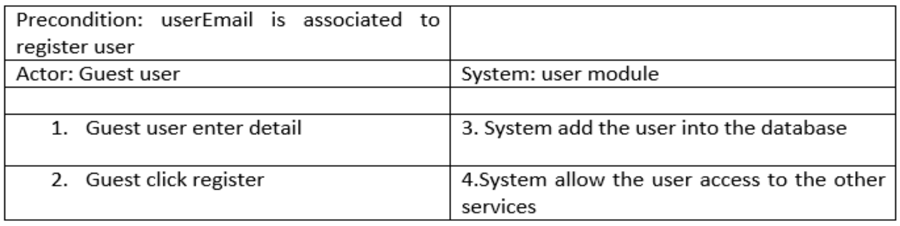


Figure 2 class diagram.

8.1.2 Precondition Table

CU1 **getUserEmail(email)**



Precondition: userEmail is associated to register user	
Actor: Guest user	System: user module
1. Guest user enter detail	3. System add the user into the database
2. Guest click register	4.System allow the user access to the other services

Figure 3 class diagram.

8.2 Service contracts

The following services should be provided

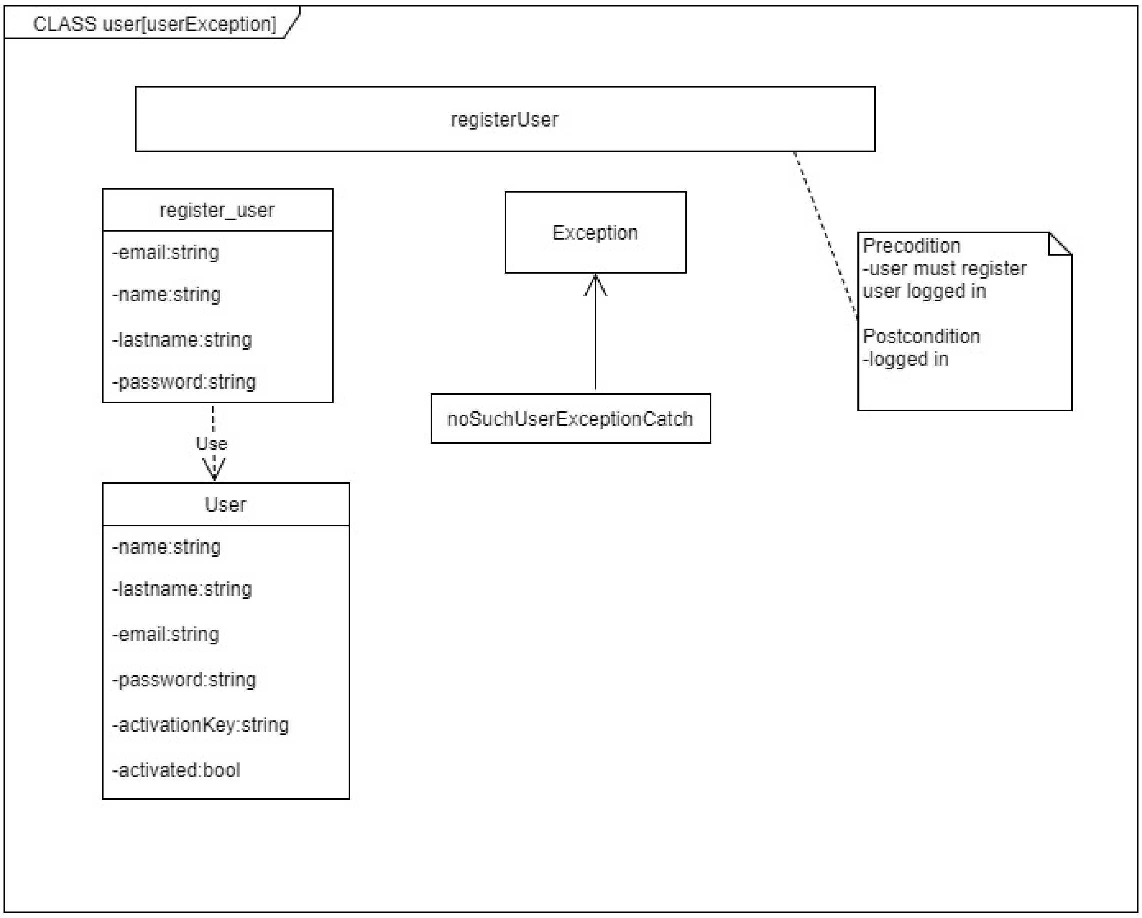


Figure 4 class diagram.

9 Display module

9.1 Scope

The display module construts graphs, charts and tables to represent the benchmark results, and how different algorithms compare against one another in terms of perfolrmance.

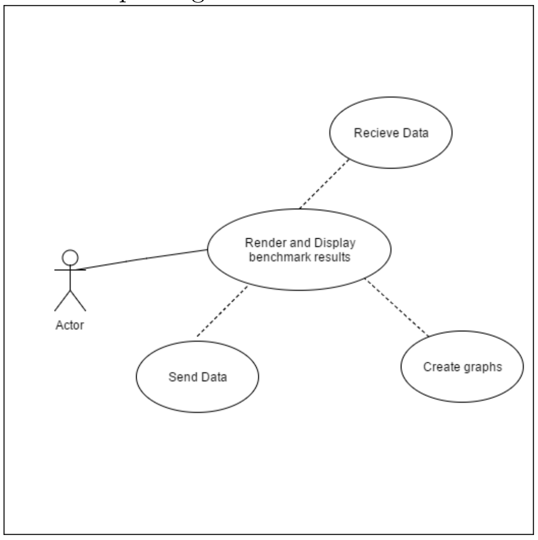


Figure 5 use case.

The display module concerns itself with rendering and displaying the results of benchmarks. The display module concerns itself primarily about creating graphs, charts and tables to represent the benchmarking results in a easily readable manner.

9.1.1 Domain model

The Display domain model is shown in Figure 6

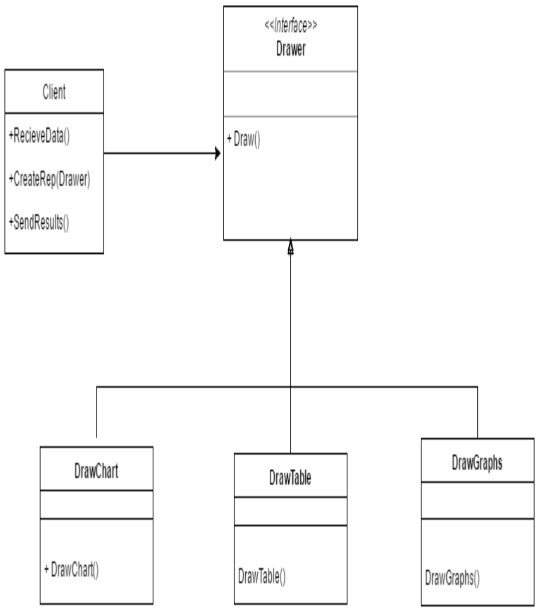
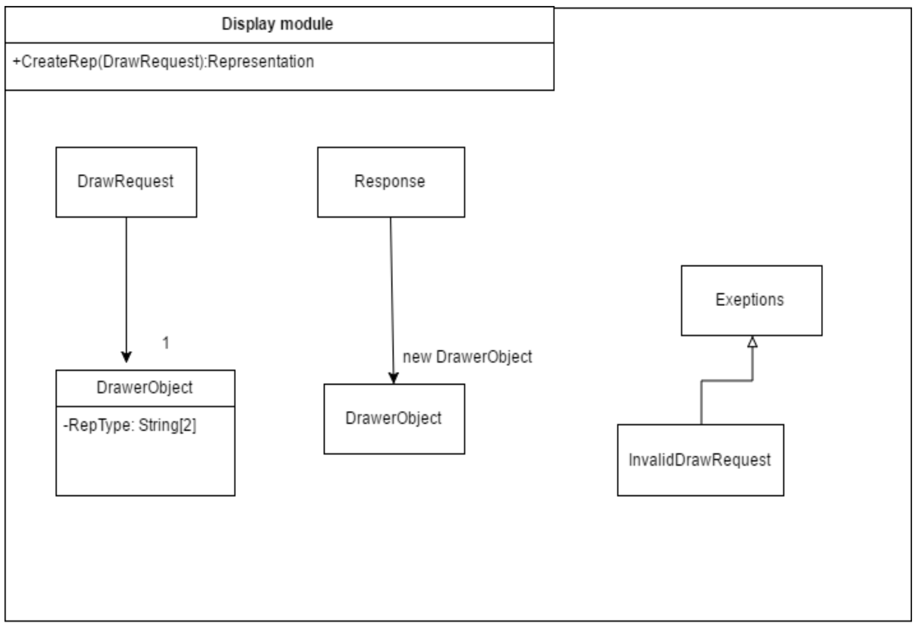


Figure 6 class diagram.
The domain model indicates how the client will use the Drawer to create the specific representation of the benchmark results.

9.2 Service contracts

The following services should be provided



<<PreConditions>>
1. The Draw request is validated and complete
<<PostConditions>>
1. The DrawObject is persisted in the system

Figure 7 Service Contracts.
The service contract specified above will be refused under one condition. If the draw request fails the validation protocol, then the contract can be refused. Otherwise, the DrawObject will be persisted and stored in the system.

9.3 Technologies

When it comes to Display technologies, one in particular is of note. JFreeChart - A free 100JFreeChart's extensive feature set includes:

1.A consistent and well-documented API, supporting a wide range of chart types;

2.A flexible design that is easy to extend, and targets both server-side and client-side applications;

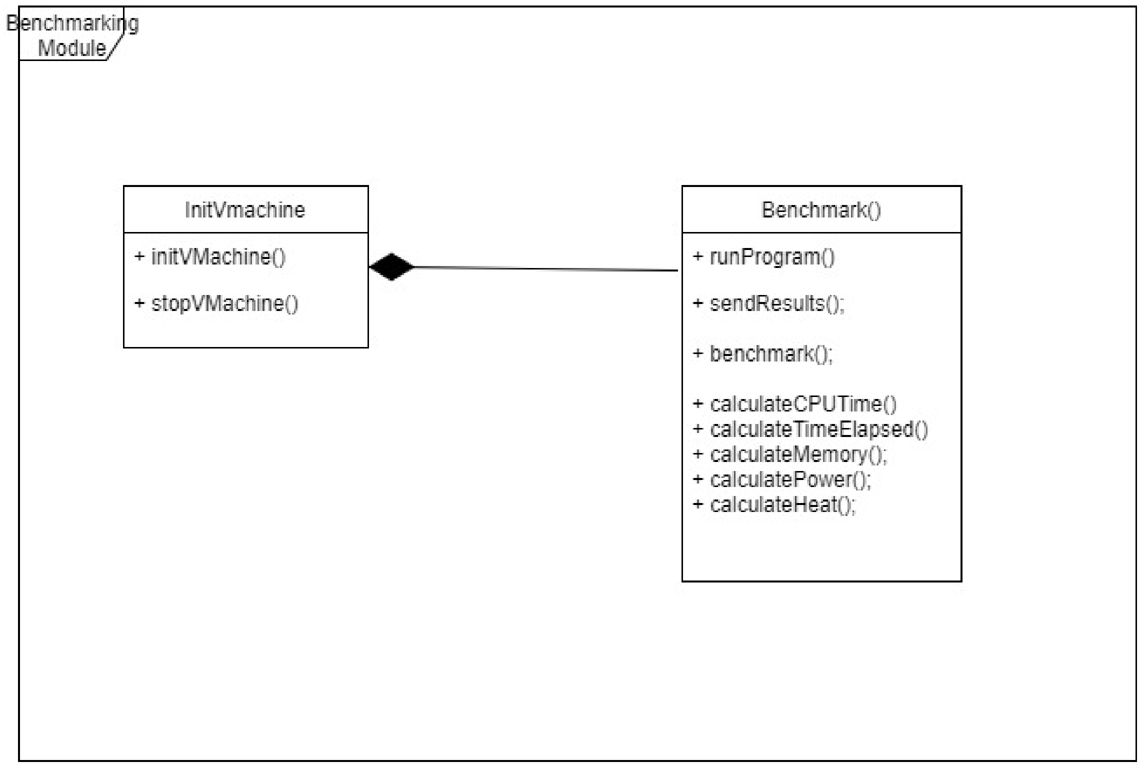
3.Support for many output types, including Swing and JavaFX components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG);

4.JFreeChart is open source or, more specifically, free software. It is distributed under the terms of the GNU Lesser General Public Licence (LGPL), which permits use in proprietary applications.

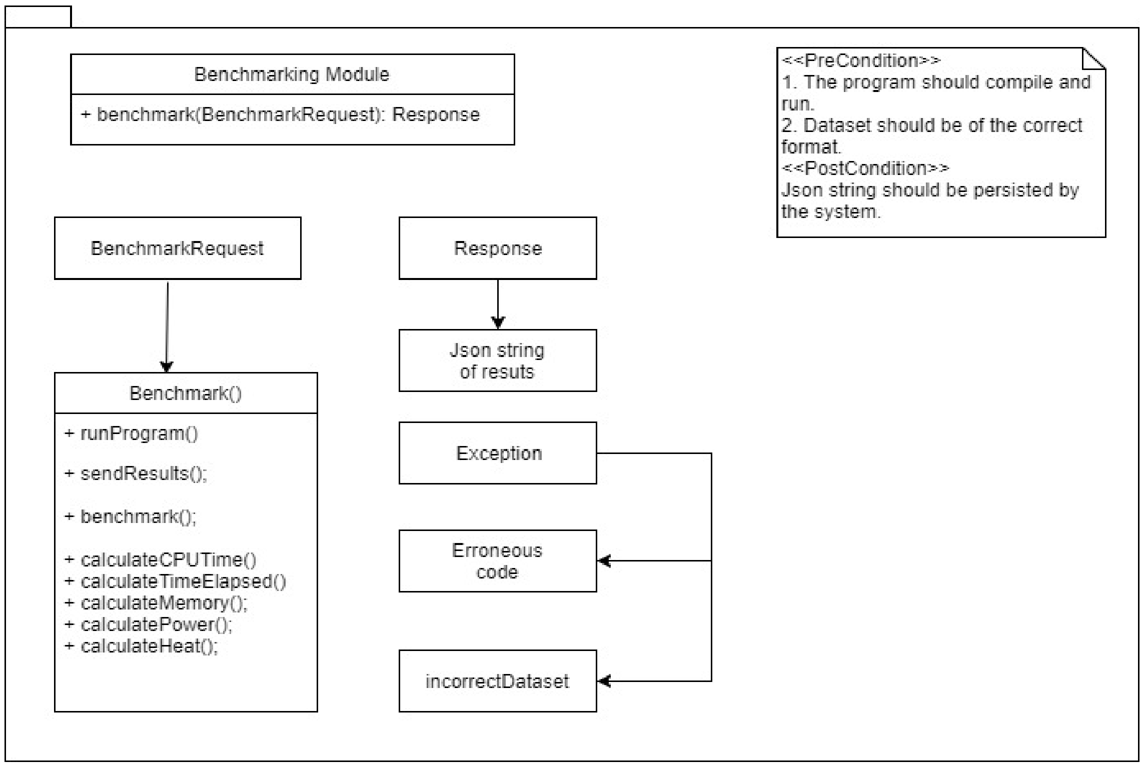
10 Bench-marking and VM initializing Module

10.1 Scope

The module initializes and deploys VMs based on the request(s). On the VMs we will have a Java program that runs other programs, that is compile them, run them with the user specified data sets and then benchmark the program or algorithms. This module will initialize VMs on request and can initialize them concurrently within seconds.



10.2 Service Contracts.

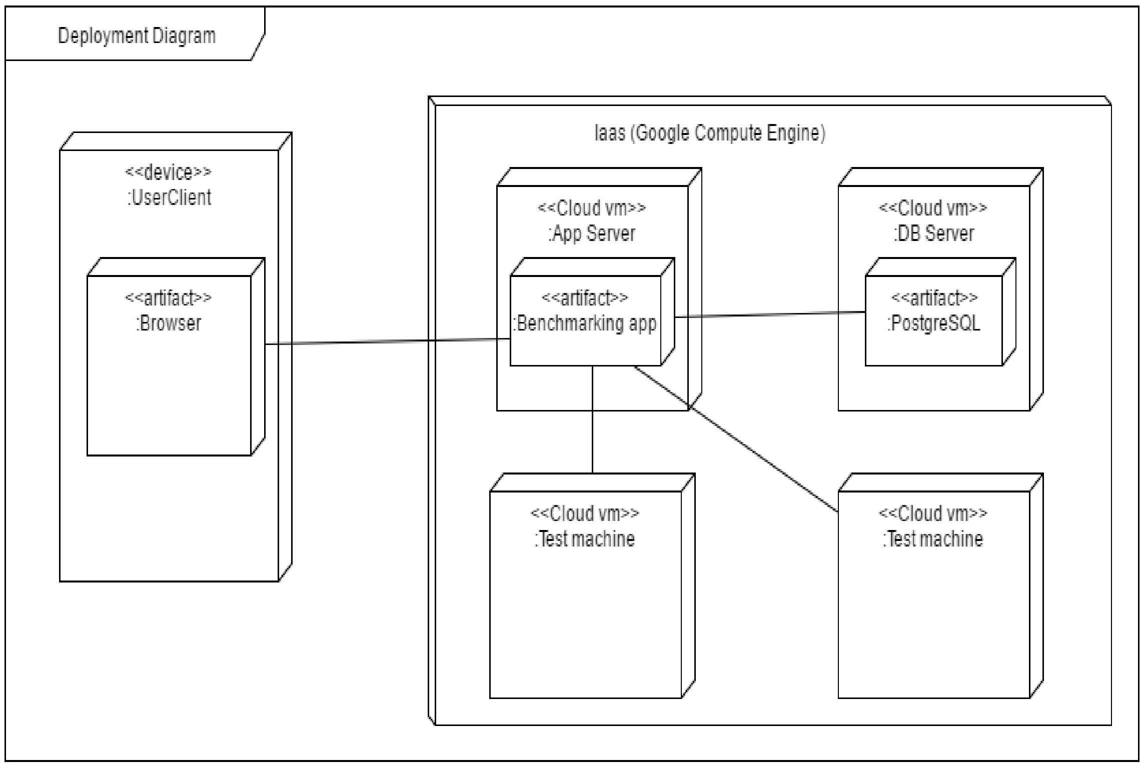


10.3 Technologies.

Osv - the open source operating system designed for the cloud. Built from the ground up for effortless deployment and management, with superior performance.

Unikernels - specialized, single address space machine images constructed by using library operating systems(Osv in our case).We select, from a modular stack, the minimal set of libraries which correspond to the OS constructs required for our application to run.

11 System Deployment Diagram


Deployment Diagram
<<device>>
:UserClient
<<artifact>>
:Browser
Iaas (Google Compute Engine)
<<Cloud vm>>
:App Server
<<artifact>>
:Benchmarking app
<<Cloud vm>>
:DB Server
<<artifact>>
:PostgreSQL
<<Cloud vm>>
:Test machine
<<Cloud vm>>
:Test machine