МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

ОБНИНСКИЙ ИНСТИТУТ АТОМНОЙ ЭНЕРГЕТИКИ

Отделение ИКС

Проектное задание

по дисциплине “Облачные сервисы и технологии”

Паттерны:

«Декоратор. Мост»

ВЫПОЛНИЛ

студент гр. ИВТ-М20 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лискунов Р. Г.

ПРОВЕРИЛ

доцент кафедры КССТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тельнов В.П.

Обнинск, 2021

Содержание

[1. Постановка задачи 4](#_Toc66640493)

[1.1 Цель работы 4](#_Toc66640494)

[1.2 Формулировка задачи 4](#_Toc66640495)

[2. Паттерны проектирования 5](#_Toc66640496)

[2.1 Декоратор 5](#_Toc66640497)

[2.1.1 Основные характеристики 5](#_Toc66640498)

[2.1.2. Решение 5](#_Toc66640499)

[2.2 Мост 6](#_Toc66640500)

[2.2.1 Основные характеристики 6](#_Toc66640501)

[2.2.2. Решение 6](#_Toc66640502)

[3. Спецификация требований 7](#_Toc66640503)

[3.1 Диаграмма совокупных требований 7](#_Toc66640504)

[3.2 Диаграмма системных требований 8](#_Toc66640505)

[3.3 Диаграмма требований к интерфейсу 9](#_Toc66640506)

[3.4 Диаграмма функциональных требований 10](#_Toc66640507)

[3.5 Прочие диаграммы требований 11](#_Toc66640508)

[3.6 Диаграмма прецедентов 12](#_Toc66640509)

[3.7 Сценарии выполнения прецедентов 13](#_Toc66640510)

[4. Технический проект 15](#_Toc66640511)

[4.1 Диаграмма классов 17](#_Toc66640512)

[4.1.1 Декоратор 17](#_Toc66640513)

[4.1.2 Мост 17](#_Toc66640514)

[4.2 Диаграммы последовательности 17](#_Toc66640515)

[4.3 Диаграммы компонентов 17](#_Toc66640516)

[4.4 Диаграммы коммуникации 17](#_Toc66640517)

[4.5 Диаграмма развёртывания 17](#_Toc66640518)

[5. Пользовательский интерфейс 18](#_Toc66640519)

[5.1 Пользователь roman 18](#_Toc66640520)

[5.2 Пользователь ubuntu 19](#_Toc66640521)

[5.3 Пользователь root 20](#_Toc66640522)

[6. Тестирование 21](#_Toc66640523)

[6.1 Сценарии тестирования, тестовые условия и наборы данных 21](#_Toc66640524)

[6.2 Протокол тестирования 23](#_Toc66640525)

[6.3 Результаты тестирования и покрытия кода 24](#_Toc66640526)

[6.4 Результаты тестирования и рефакторинга 25](#_Toc66640527)

[6.4.1 Используемые метрики процесса тестирования 25](#_Toc66640528)

[6.4.2 Рефакторинг кода 25](#_Toc66640529)

[7. Руководство пользователя 26](#_Toc66640530)

[7.1 Область применения 26](#_Toc66640531)

[7.2 Краткое описание возможностей 26](#_Toc66640532)

[7.3 Уровень подготовки пользователя 26](#_Toc66640533)

[7.4 Назначение программы 26](#_Toc66640534)

[8. Листинг кода 27](#_Toc66640535)

[9. Протокол изменений, внесенный в первоначальный проект 28](#_Toc66640536)

# 1. Постановка задачи

## 1.1 Цель работы

Требуется для двух паттернов проектирования софта (Декоратор и Мост) реализовать на языке Python работоспособное веб-приложение, размещенное на облачном хостинге. Пример должен быть сделан на языке программирования Python с использованием фреймворка Flask.

## 1.2 Формулировка задачи

Проектируемое приложение: «Система информации о состоянии сервера».

Поддерживаемые данные

* Общие данные
  + Имя сервера
  + Платформа
  + Путь запуска скрипта для анализа всех данных
  + Домашняя директория
* Состояние процессора
  + Число ядер
  + Процент нагрузки
* Состояние жестких дисков
  + Процент загрузки
  + Общее количество пространства
  + Используемое количество пространства

Поддерживаемые операции

* Получение списка показателей, в том числе в текущий промежуток времени.
* Получение списка директорий, в том числе домашняя и текущая.
* Получение числа ядер процессора сервера.
* Получение процента загрузки жестких дисков на сервере.
* Математическое определение процента нагрузки на сервер.
* Математическое определение используемого количество пространства на жестком диске.

# 2. Паттерны проектирования

## 2.1 Декоратор

Декоратор — это структурный паттерн проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Шаблон Декоратор предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности.

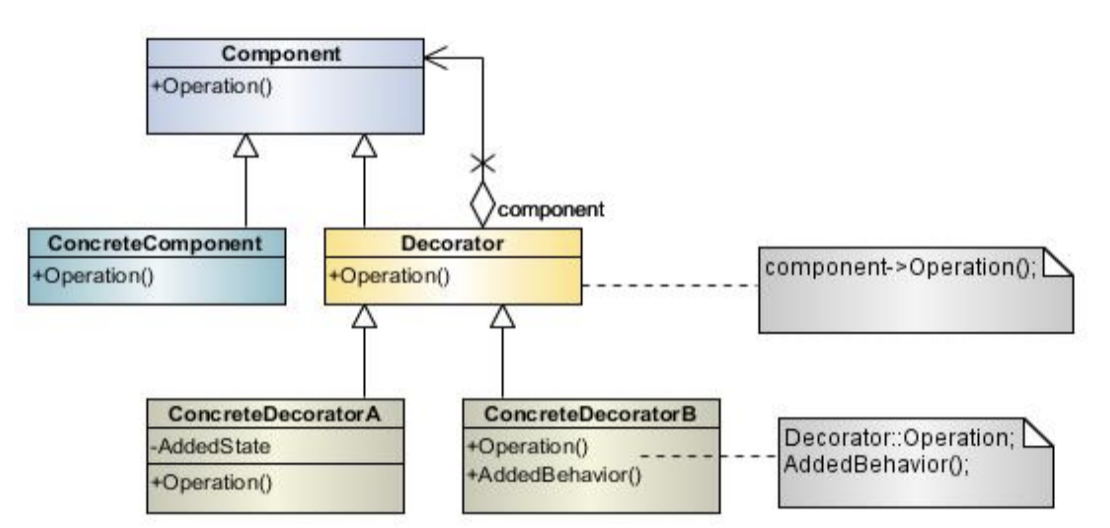
### 2.1.1 Основные характеристики

**Задача**. Объект, который предполагается использовать, выполняет основные функции. Однако может потребоваться добавить к нему некоторую дополнительную функциональность, которая будет выполняться до, после или даже вместо основной функциональности объекта.

**Способ решения**. Декоратор предусматривает расширение функциональности объекта без определения подклассов.

**Реализация**. Создаётся абстрактный класс, представляющий как исходный класс, так и новые, добавляемые в класс функции. В классах-декораторах новые функции вызываются в требуемой последовательности — до или после вызова последующего объекта.

### 2.1.2. Решение



## 2.2 Мост

Мост — это структурный паттерн проектирования, который разделяет один или несколько классов на две отдельные иерархии — абстракцию и реализацию, позволяя изменять их независимо друг от друга. Паттерн Мост использует инкапсуляцию, агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами.

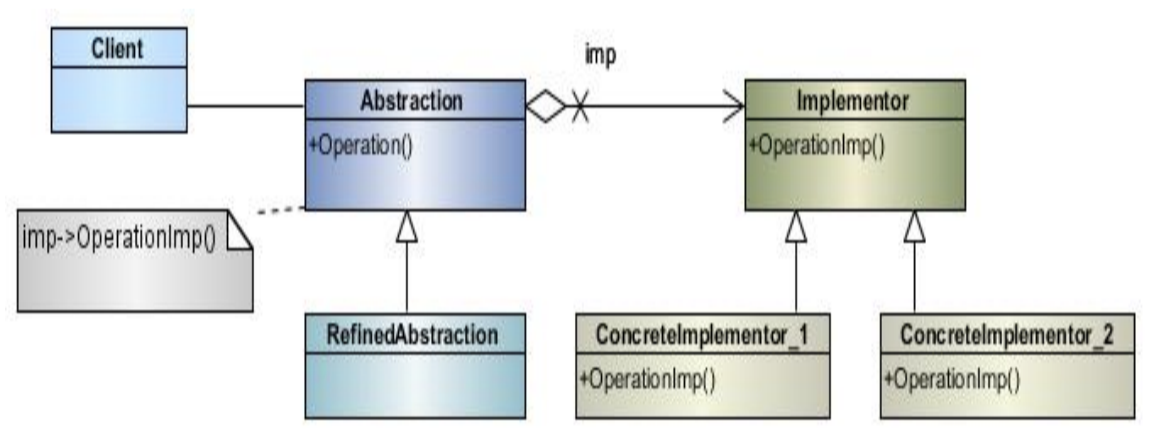
### 2.2.1 Основные характеристики

**Задача**. При частом изменении класса преимущества объектно-ориентированного подхода становятся очень полезными, позволяя делать изменения в программе, обладая минимальными сведениями о реализации программы. Шаблон Мост является полезным там, где часто меняется не только сам класс, но и то, что он делает.

**Способ решения**. Когда абстракция и реализация разделены, они могут изменяться независимо. Другими словами, при реализации через паттерн Мост, изменение структуры интерфейса не мешает изменению структуры реализации.

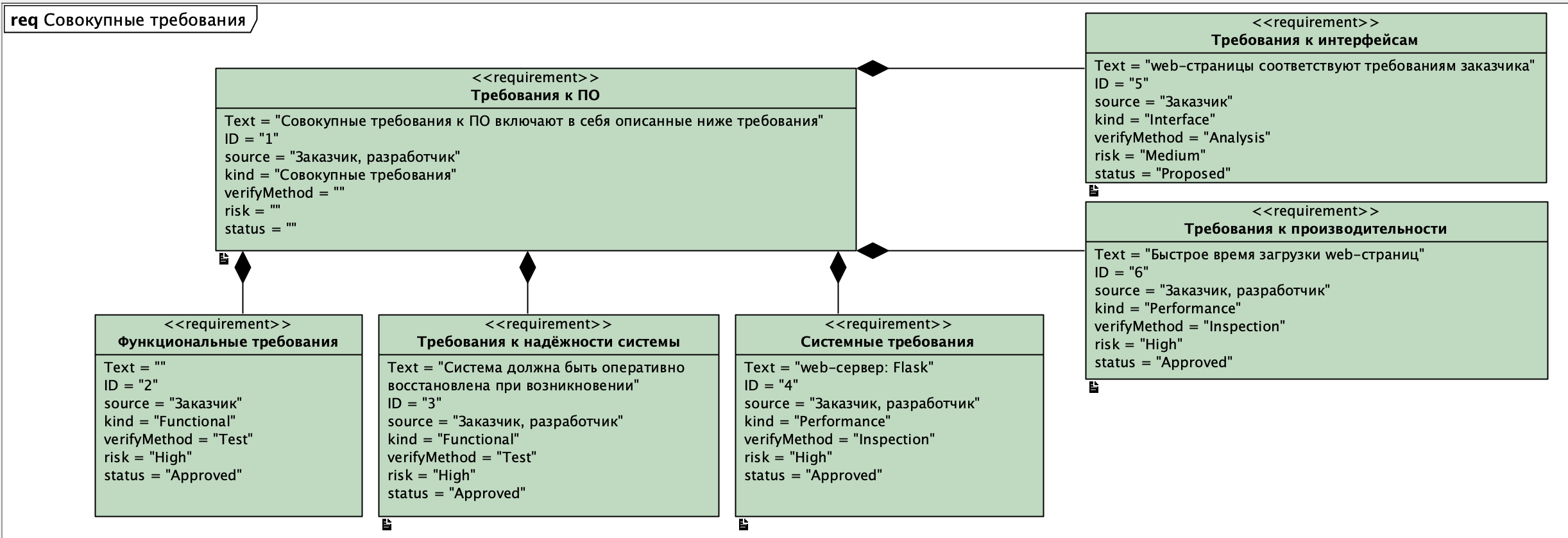
**Реализация**. Паттерн Мост предлагает выделить одну из таких плоскостей в отдельную иерархию классов, храня ссылку на один из её объектов в первоначальном классе.

### 2.2.2. Решение

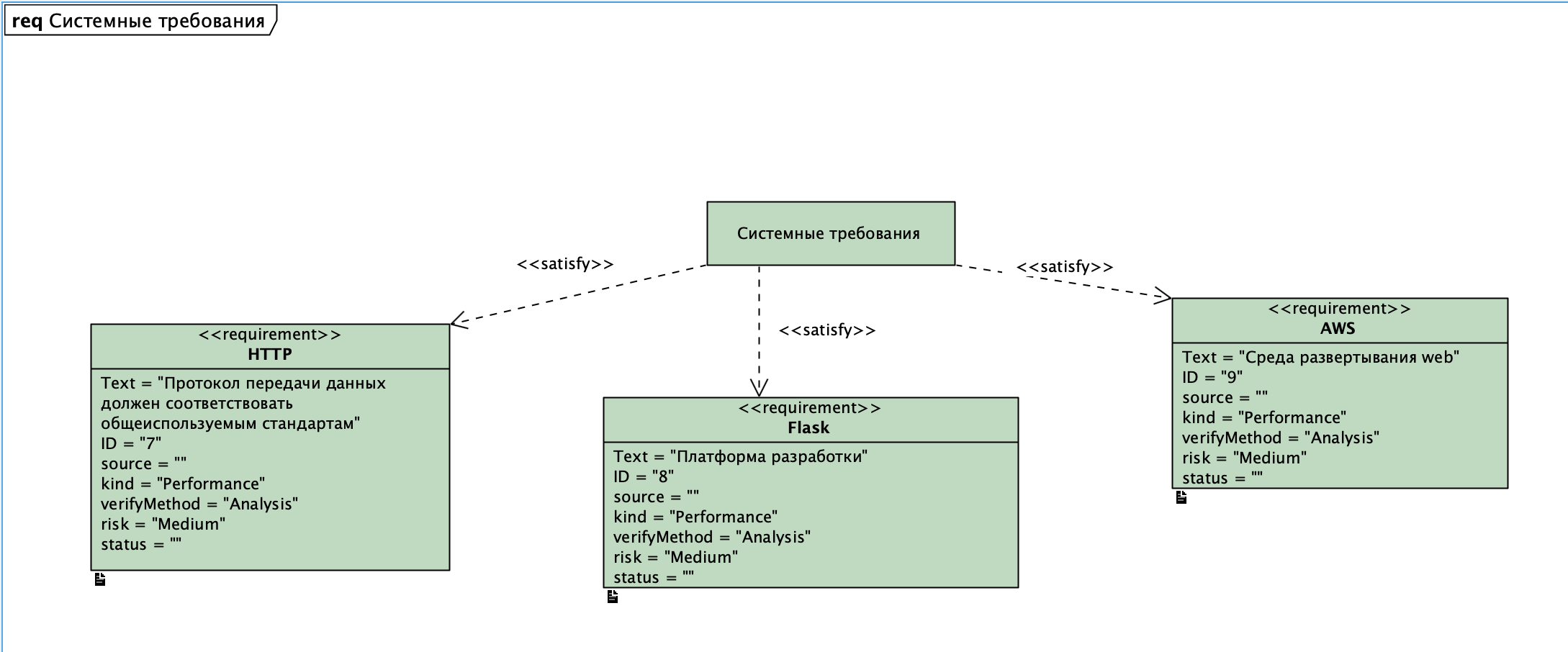


# 3. Спецификация требований

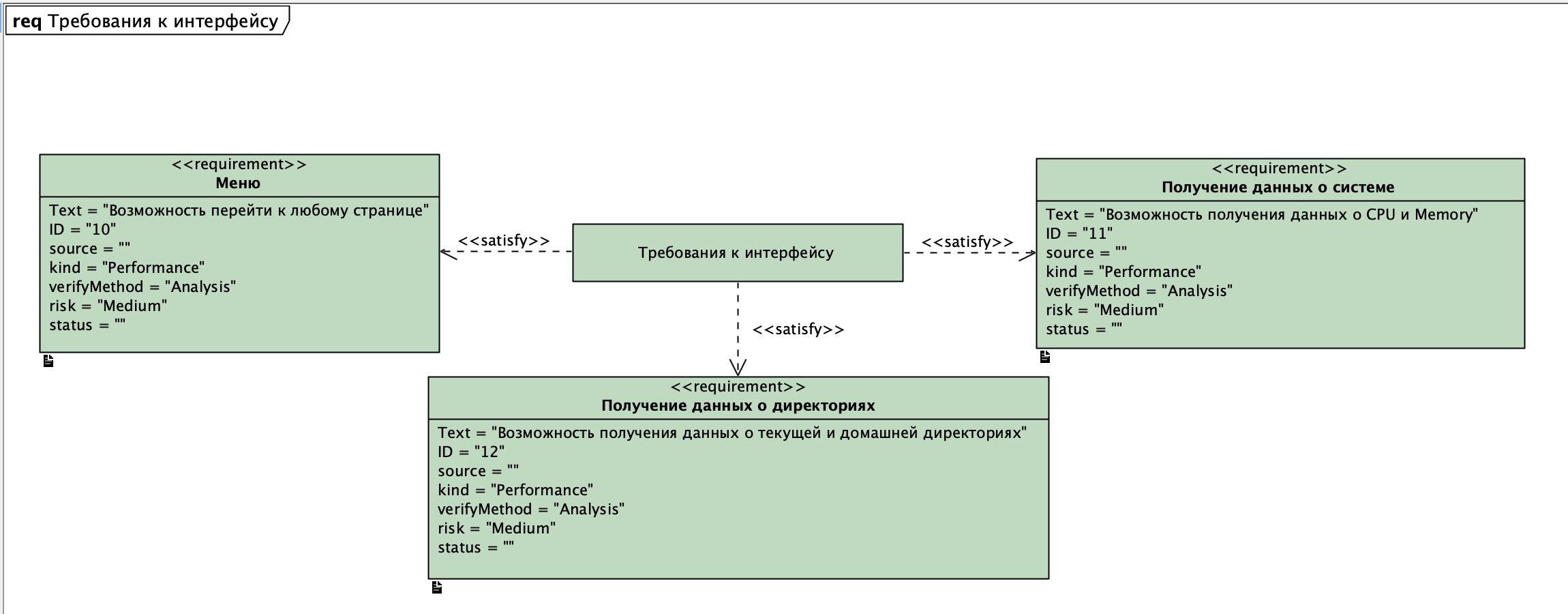
## 3.1 Диаграмма совокупных требований



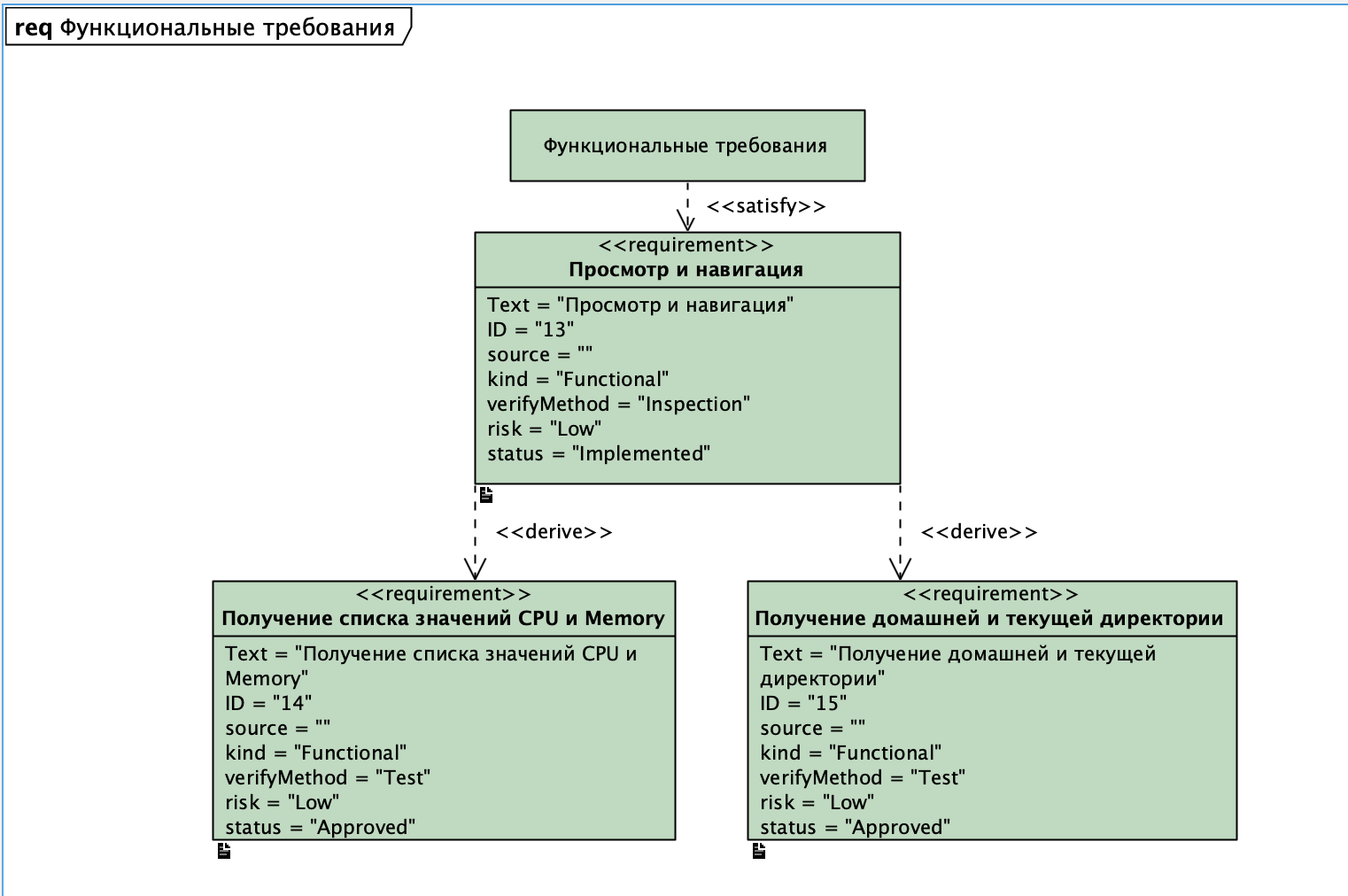
## 3.2 Диаграмма системных требований



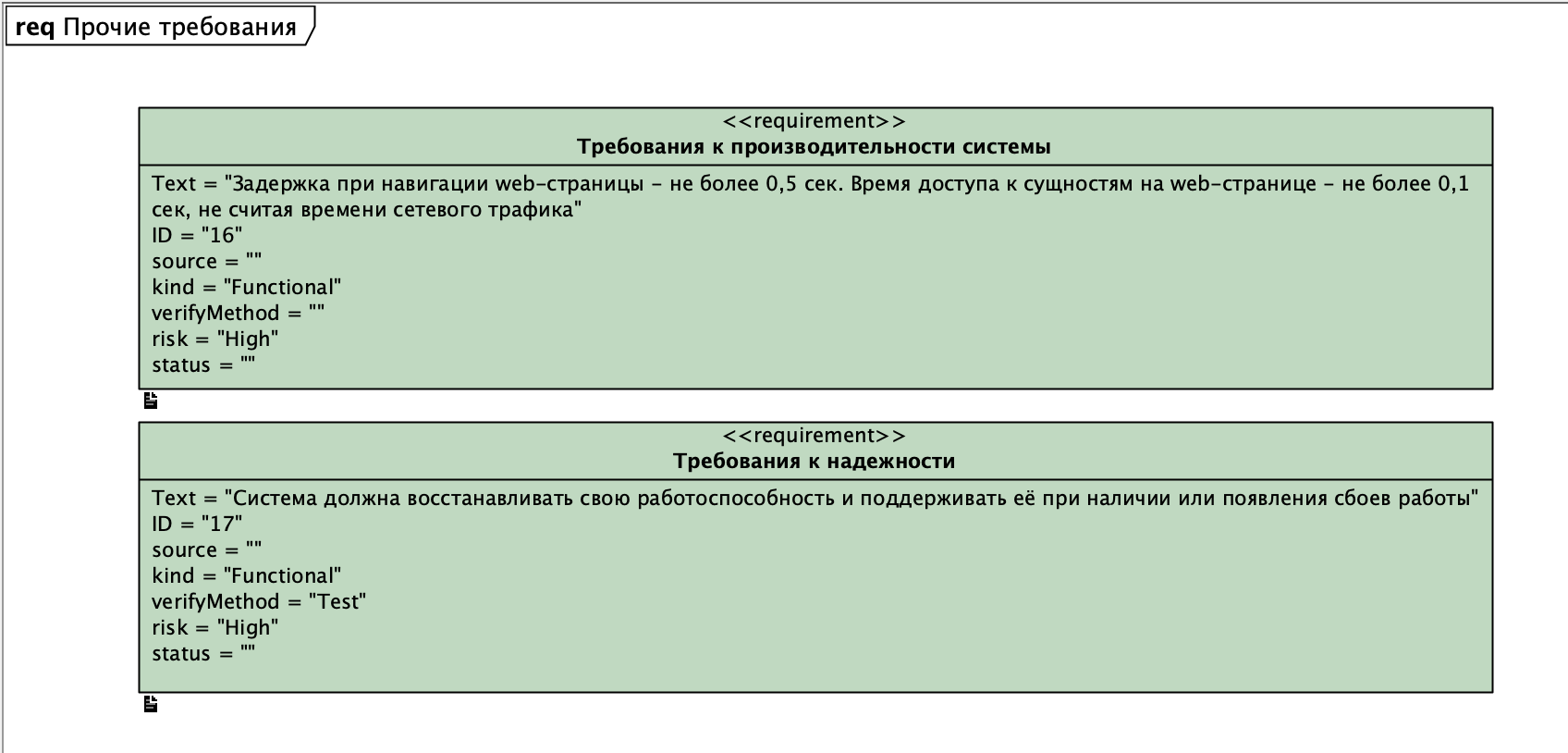
## 3.3 Диаграмма требований к интерфейсу



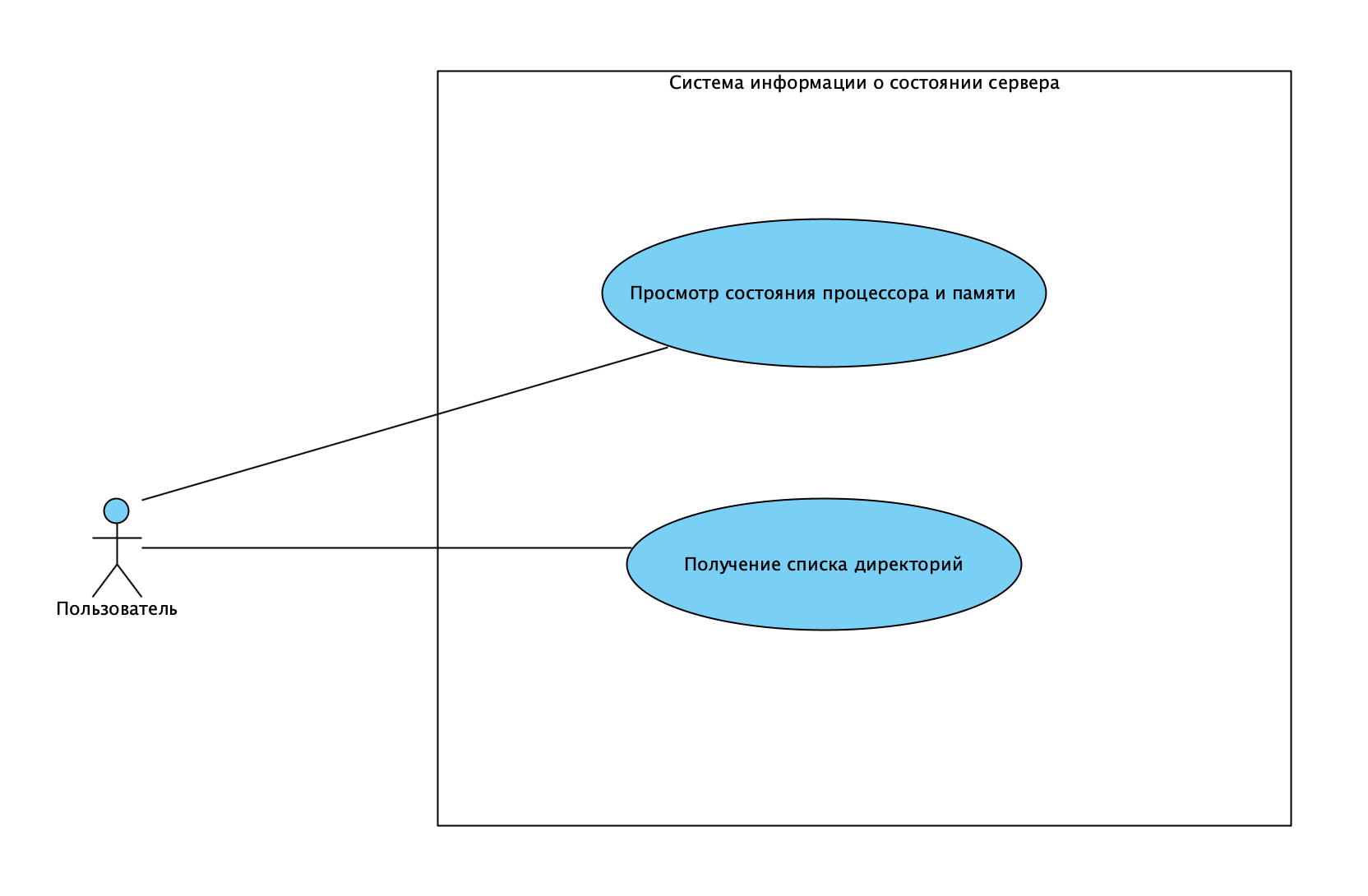
## 3.4 Диаграмма функциональных требований



## 3.5 Прочие диаграммы требований



## 3.6 Диаграмма прецедентов



Пользователь, зайдя в систему может проверить состояние своего сервера, получить детализацию данных об имени компьютера, процентного соотношения загрузки ядра и загрузки оперативной памяти. Дополнительно пользователю доступна возможность получить список директорий на данном сервере: домашняя, чтобы определить от какого конкретно пользователя была запущена система, и текущая, чтобы определить место запуска данного скрипта мониторинга.

## 3.7 Сценарии выполнения прецедентов

№1 Раздел «Общее описание сценария»

|  |  |
| --- | --- |
| Прецедент | Получение состояния процессора и памяти |
| Акторы | Пользователь |
| Цель | Просмотр пользователем мониторинга информации о процессоре и памяти |
| Краткое описание | Пользователь отправляет на сервер соответствующий запрос |
| Тип | Базовый прецедент |
| Ссылки на другие прецеденты | - |

Раздел «Типичный ход событий»

|  |  |
| --- | --- |
| Действия акторов | Отклик системы |
| 1. Пользователь, находясь в системе отправляет запрос на получение информации | 2. Система отображает имя сервера, нагрузку на процессор и память |

№2 Раздел «Общее описание сценария»

|  |  |
| --- | --- |
| Прецедент | Получение списка директорий |
| Акторы | Пользователь |
| Цель | Просмотр пользователем домашней и текущей директории |
| Краткое описание | Пользователь отправляет на сервер соответствующий запрос |
| Тип | Базовый прецедент |
| Ссылки на другие прецеденты | - |

Раздел «Типичный ход событий»

|  |  |
| --- | --- |
| Действия акторов | Отклик системы |
| 1. Пользователь, находясь в системе отправляет запрос на получение информации | 2. Система отображает имя сервера, нагрузку на процессор и память |

# 4. Технический проект

Классы: CPU (Процессор), Memory (Память), Windows, Mac, HomeDirectory (Домашняя директория), CurrentDirectory (Текущая директория). Опишем каждый класс отдельно.

Класс «Процессор» будет содержать в себе следующие поля:

1. **Ядра** – поле private типа “Integer”. Назначение: число ядер сервера.
2. **Нагрузка** – поле private типа “Float”. Назначение: процентная нагрузка на сервер.
3. **Методы**
   1. **демонстрация() –** вернуть текущие значения за одну секунду.

Класс «Память» будет содержать в себе следующие поля:

1. **Всего** – поле private типа “Float”. Назначение: число единиц памяти в мегабайтах.
2. **Используемо** – поле private типа “Float”. Назначение: используемое число единиц памяти в мегабайтах.
3. **Нагрузка** **–** поле private типа “Float”. Назначение: процентная нагрузка на сервер.
4. Методы
   1. **демонстрация() –** вернуть текущие значения за одну секунду.

Класс «Windows» будет содержать в себе следующие поля:

1. Методы
   1. **операция() –** вернуть текущие значение.

Класс «Mac» будет содержать в себе следующие поля:

1. Методы
   1. **операция() –** вернуть текущие значение.

Класс «Домашняя директория» будет содержать в себе следующие поля:

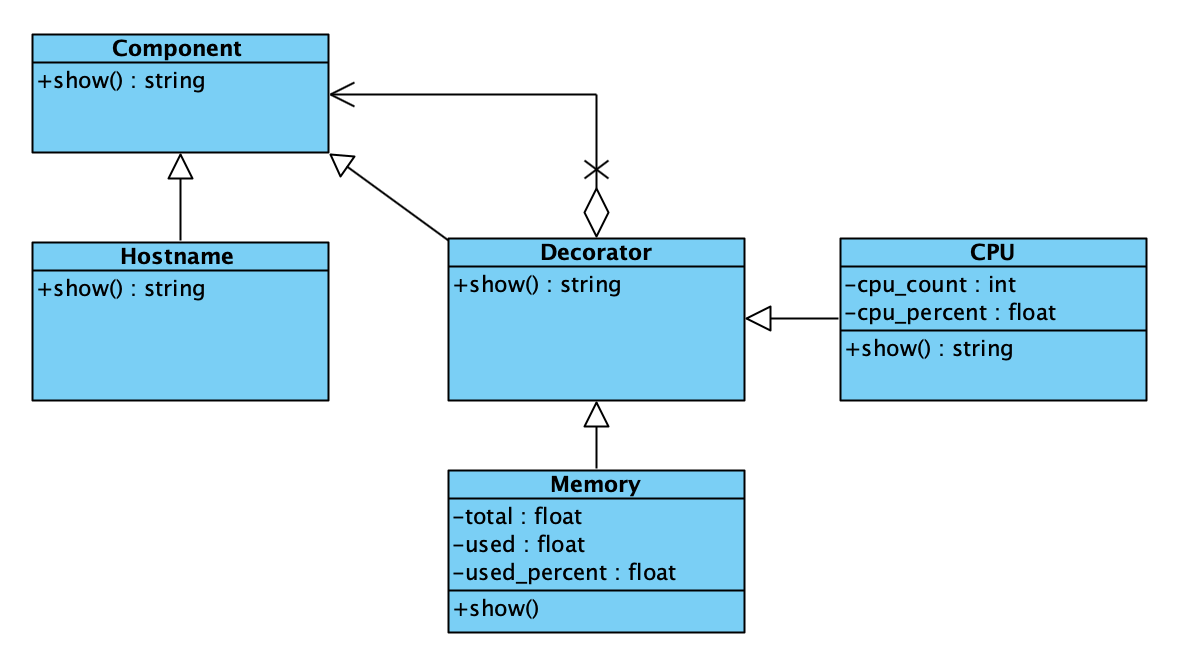
1. **Директория** –– поле private типа “String”. Назначение: полный путь до текущего пользователя сервера.
2. Методы
   1. **операция() –** вернуть текущие значение домашней директории.

Класс «Текущая директория» будет содержать в себе следующие поля:

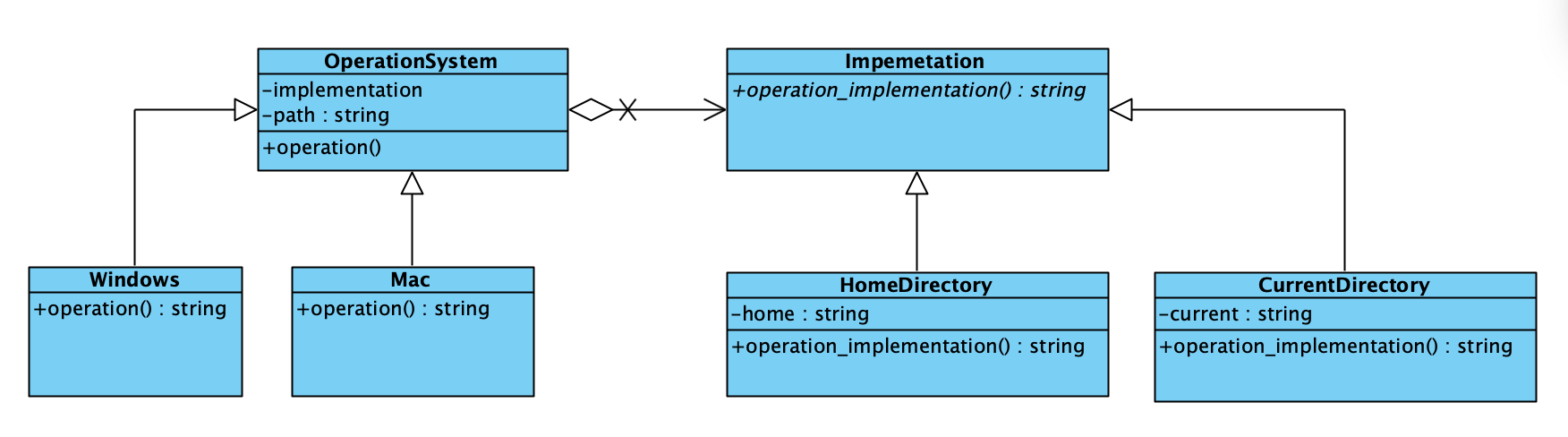
1. **Директория** –– поле private типа “ String”. Назначение: полный путь до исполняемого модуля.
2. Методы
   1. **операция() –** вернуть текущие значение текущей директории.

## 4.1 Диаграмма классов

### 4.1.1 Декоратор



### 4.1.2 Мост



## 4.2 Диаграммы последовательности

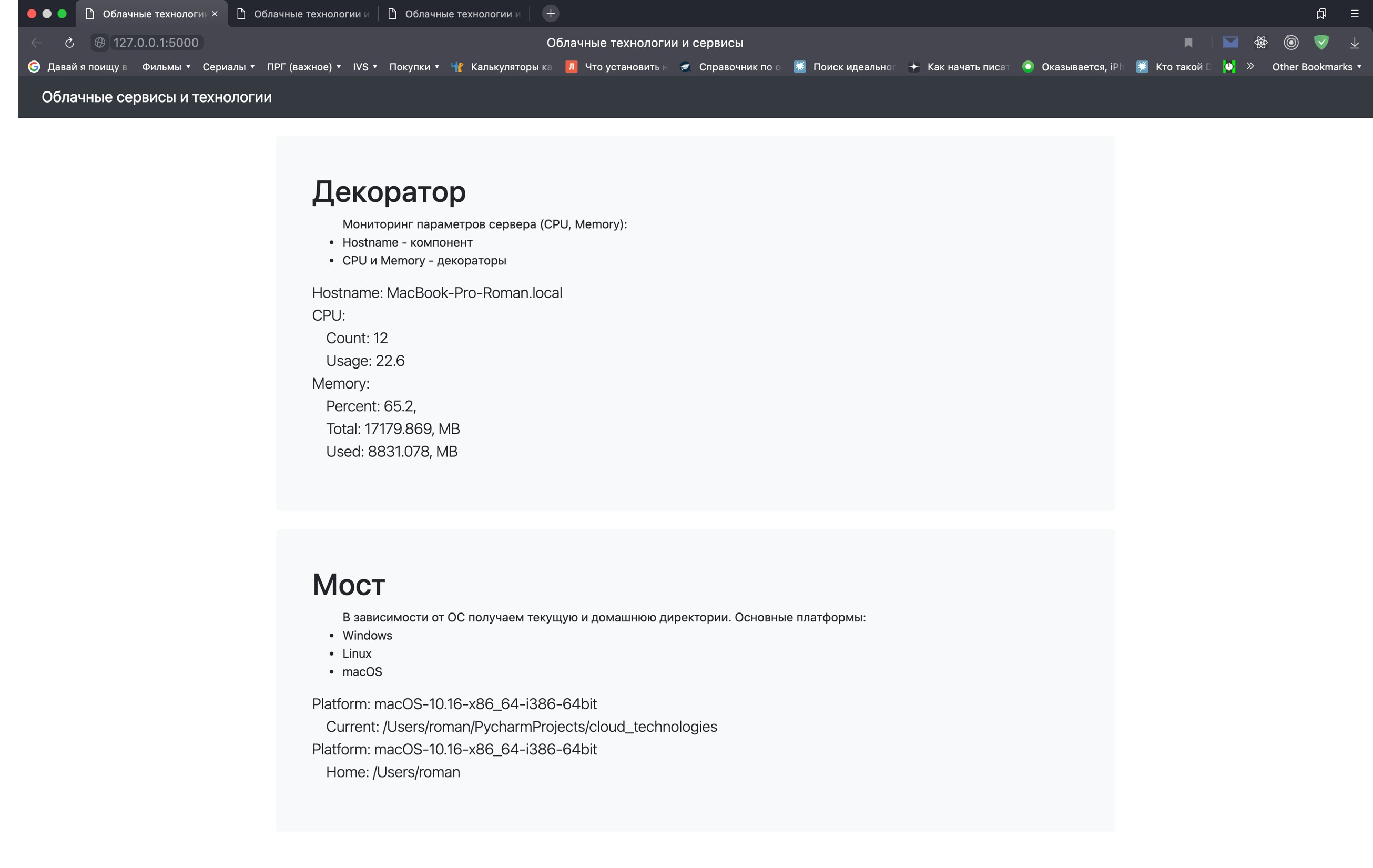
## 4.3 Диаграммы компонентов

## 4.4 Диаграммы коммуникации

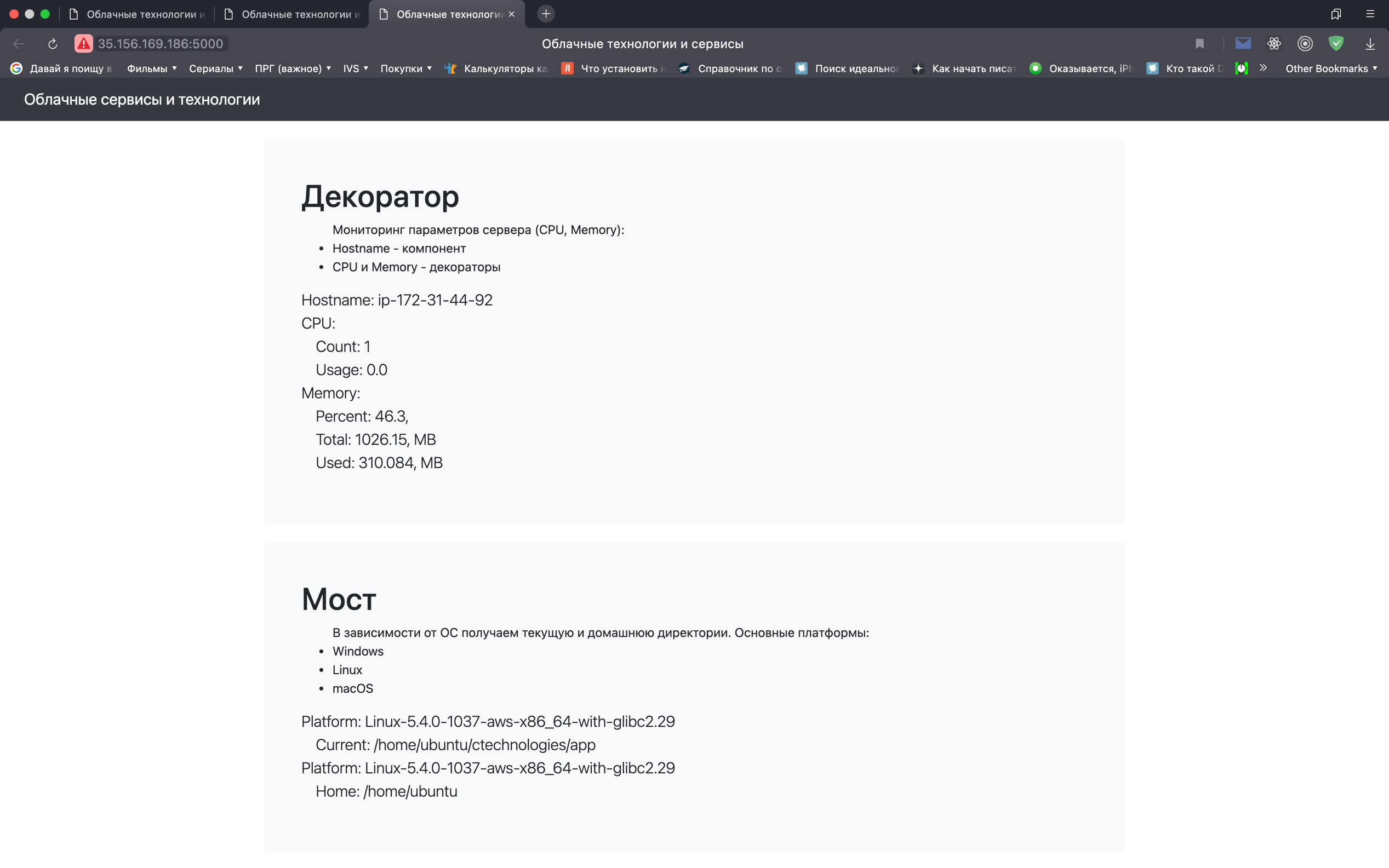
## 4.5 Диаграмма развёртывания

# 5. Пользовательский интерфейс

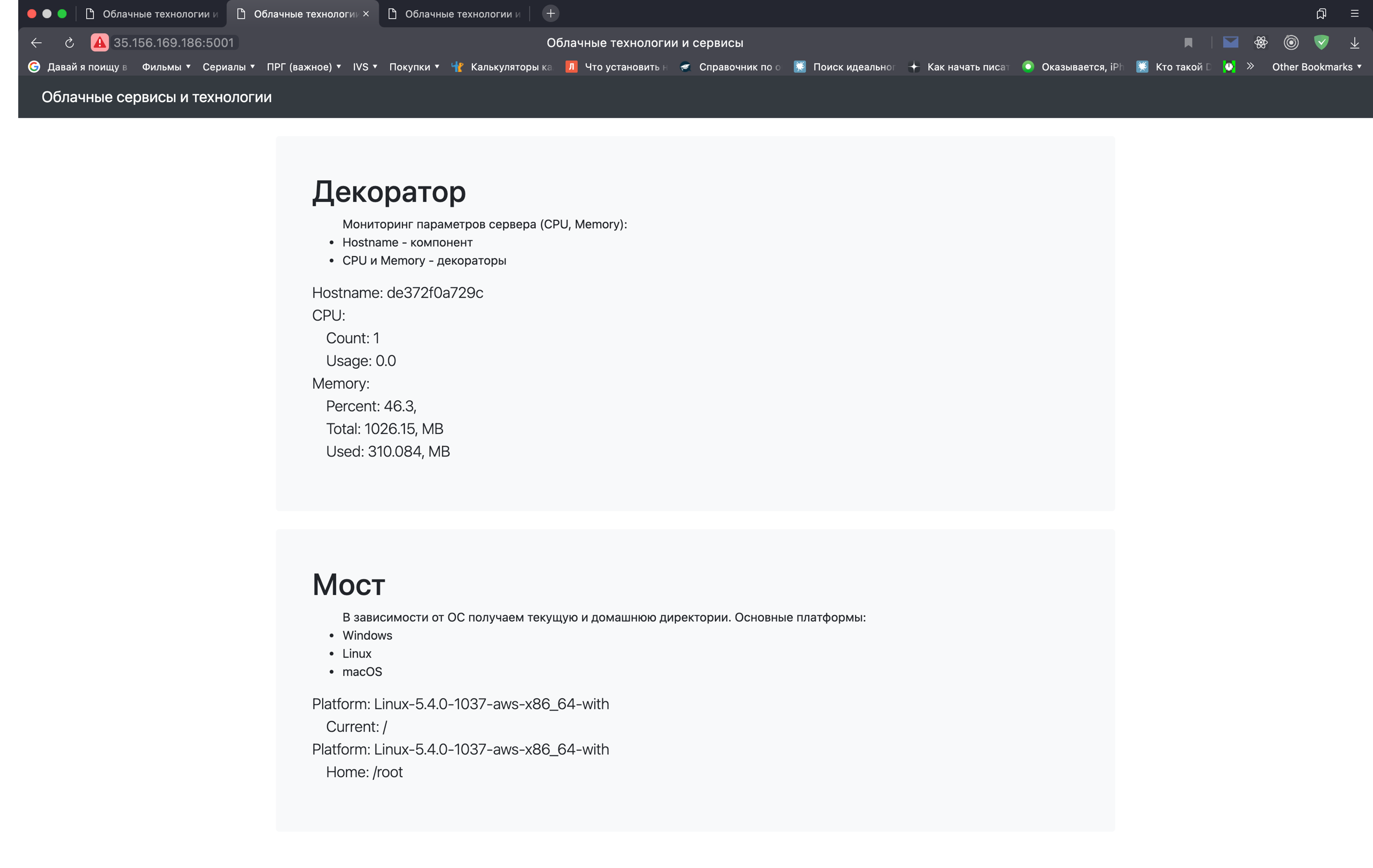
# 5.1 Пользователь roman



## 5.2 Пользователь ubuntu



## 5.3 Пользователь root



# 6. Тестирование

Выполнить комплексное автоматизированное тестирование программного решения, которое было разработано в предыдущих лабораторных работах.

Выполнить расчет метрик тестирования. Метрики процесса тестирования по стандарту IEEE 982.1-1998

Продумать варианты улучшения внешнего вида кода, компоновки папок проекта, оптимизации программного решения (рефакторинг).

## 6.1 Сценарии тестирования, тестовые условия и наборы данных

I. На основании pytest-тестов проверить работоспособность написанных классов по отдельности

1. Класс CurrentDirectory.py

* Проверка отображение текущей директории методом operation\_implementation();

1. Класс HomeDirectory.py

* Проверка отображение домашней директории методом operation\_implementation();

1. Класс Mac.py

* Проверка отображение платформы методом operation();

1. Класс Windows.py

* Проверка отображение платформы методом operation();

1. Класс CPU.py

* Проверка отображения числа ядер и нагрузки на процессор методом show();

1. Класс Memory.py

* Проверка отображения числа общей и используемой памяти вместе с нагрузкой на методом show();

II. Через веб-интерфейс протестировать работоспособность проекта в целом

1. Проверить на орфографические и пунктуационные ошибки.
2. Проверить визуальное отображение элементов.
3. Проверить правильность переходов по страницам приложения.

III. Протестировать работоспособность на разных компьютерах и браузерах

1. Проверить работу в различных браузерах

* Internet Explorer
* Google Chrome
* Mozilla Firefox
* Opera

2. Проверить быстродействие на разных компьютерах

* Процессор 8 ядер 3.2 GHz, 8 Gb оперативной памяти
* Процессор 2 ядра 1.80 GHz, 4 Gb оперативной памяти

## 6.2 Протокол тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Класс** | **Наименование тестирования** | **Ожидаемый результат** | **Результаты теста** |
| 1 | CurrentDirectory.py | Отображение текущей директории | Директория отображена | Успешен |
| 2 | HomeDirectory.py | Отображение домашней директории | Директория отображена | Успешен |
| 3 | Windows.py | Отображение платформы Windows | Платформа отображена | Успешен |
| 4 | Mac.py | Отображение платформы Darwin | Отображение платформы Windows | Успешен |
| 5 | CPU.py | Отображение данных о процессоре | Метрики отображены | Успешен |
| 6 | Memory.py | Отображение данных о памяти | Метрики отображены | Успешен |

## 6.3 Результаты тестирования и покрытия кода

## 6.4 Результаты тестирования и рефакторинга

## 6.4.1 Используемые метрики процесса тестирования

1. **IEEE 1**. Плотность отказов = [Число уникальных отказов, найденных при тестировании] / [Число строк кода]:

2. **IEEE 18**. Надёжность работы выражается вероятностью того, что в k произвольных случаях работы программа вернёт корректный результат. Эта величина оценивается через выполнение некоторого числа запусков программы (N) и вычисление числа случаев успешной работы (S). Вероятность успеха, таким образом, вычисляется как S/N, а вероятность возможности отработать k раз успешно – как произведение вероятностей каждого успешного запуска, то есть [S/N]\*[ S/N]\*…\*[ S/N], или [S/N]^k. Входные данные для каждого запуска выбираются произвольно и независимо от предыдущего запуска.

3. **IEEE 22**. Оценка числа оставшихся отказов (методом засева). Эта оценка получена путем «засеивания» в программу N произвольных отказов. Если s - число найденных засеянных отказов, а f - число других отказов, найденных за тот же период тестирования, оценка равна f × N / s.

4. **IEEE 36**. Точность тестирования = f/N, где N – это число засеянных отказов, а f – это число засеянных ошибок, найденных во время тестирования.

### 6.4.2 Рефакторинг кода

Функционал определения данных о процессоре и памяти были вынесены в отдельный модуль. Большая часть способов получения данных метрик была переписана с использованием встроенных средств языка. В случае перехода на несуществующую страницу с использованием метода Error() было добавлено логирование адреса перехода.

# 7. Руководство пользователя

## 7.1 Область применения

## 7.2 Краткое описание возможностей

## 7.3 Уровень подготовки пользователя

## 7.4 Назначение программы

# 8. Листинг кода

# 9. Протокол изменений, внесенный в первоначальный проект