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](#_Toc69578233)

[1.1 Цель работы 4](#_Toc69578234)

[1.2 Формулировка задачи 4](#_Toc69578235)

[2. Паттерны проектирования 5](#_Toc69578236)

[2.1 Декоратор 5](#_Toc69578237)

[2.1.1 Основные характеристики 5](#_Toc69578238)

[2.1.2. Решение 5](#_Toc69578239)

[2.2 Мост 6](#_Toc69578240)

[2.2.1 Основные характеристики 6](#_Toc69578241)

[2.2.2. Решение 6](#_Toc69578242)

[3. Спецификация требований 7](#_Toc69578243)

[3.1 Диаграмма совокупных требований 7](#_Toc69578244)

[3.2 Диаграмма системных требований 8](#_Toc69578245)

[3.3 Диаграмма требований к интерфейсу 9](#_Toc69578246)

[3.4 Диаграмма функциональных требований 10](#_Toc69578247)

[3.5 Прочие диаграммы требований 11](#_Toc69578248)

[3.6 Диаграмма прецедентов 12](#_Toc69578249)

[3.7 Сценарии выполнения прецедентов 13](#_Toc69578250)

[4. Технический проект 15](#_Toc69578251)

[4.1 Диаграмма классов 17](#_Toc69578252)

[4.1.1 Декоратор 17](#_Toc69578253)

[4.1.2 Мост 18](#_Toc69578254)

[4.2 Диаграммы последовательности 19](#_Toc69578255)

[4.2.1 Получение информации 19](#_Toc69578256)

[4.2.2 Получение директорий 20](#_Toc69578257)

[4.2.3 Получение метрик 21](#_Toc69578258)

[4.3 Диаграммы компонентов 22](#_Toc69578259)

[4.4 Диаграммы коммуникации 23](#_Toc69578260)

[4.4.1 Получение информации 23](#_Toc69578261)

[4.4.2 Получение директорий 24](#_Toc69578262)

[4.2.3 Получение метрик 25](#_Toc69578263)

[4.5 Диаграмма развёртывания 26](#_Toc69578264)

[5. Пользовательский интерфейс 27](#_Toc69578265)

[5.1 Пользователь roman 27](#_Toc69578266)

[5.1.1 Декоратор 27](#_Toc69578267)

[5.1.2 Мост 27](#_Toc69578268)

[5.2 Пользователь ubuntu 28](#_Toc69578269)

[5.2.1 Декоратор 28](#_Toc69578270)

[5.2.2 Мост 28](#_Toc69578271)

[5.3 Пользователь root 29](#_Toc69578272)

[5.3.1 Декоратор 29](#_Toc69578273)

[5.3.2 Мост 29](#_Toc69578274)

[6. Тестирование 30](#_Toc69578275)

[6.1 Сценарии тестирования, тестовые условия и наборы данных 30](#_Toc69578276)

[6.2 Протокол тестирования 32](#_Toc69578277)

[6.3 Результаты тестирования и рефакторинга 33](#_Toc69578278)

[6.3.1 Используемые метрики процесса тестирования 33](#_Toc69578279)

[6.3.2 Рефакторинг кода 33](#_Toc69578280)

[7. Руководство пользователя 34](#_Toc69578281)

[7.1 Область применения 34](#_Toc69578282)

[7.2 Краткое описание возможностей 34](#_Toc69578283)

[7.3 Уровень подготовки пользователя 35](#_Toc69578284)

[7.4 Назначение программы 35](#_Toc69578285)

[7.4.1 Виды деятельности, функции 35](#_Toc69578286)

[7.4.2 Программные и аппаратные требования к системе 35](#_Toc69578287)

[7.4.3 Подготовка к работе 35](#_Toc69578288)

[7.5 Решение известных проблем (Troubleshooting) 36](#_Toc69578289)

[8. Листинг кода 37](#_Toc69578290)

[8.1 app.py 37](#_Toc69578291)

[8.2 bridge.py 38](#_Toc69578292)

[8.3 decorator.py 40](#_Toc69578293)

[8.4 base.html 42](#_Toc69578294)

[8.5 index.html 44](#_Toc69578295)

[8.6 bridge.html 44](#_Toc69578296)

[8.7 decorator.html 44](#_Toc69578297)

[9. Протокол изменений, внесенный в первоначальный проект 46](#_Toc69578298)

# 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.

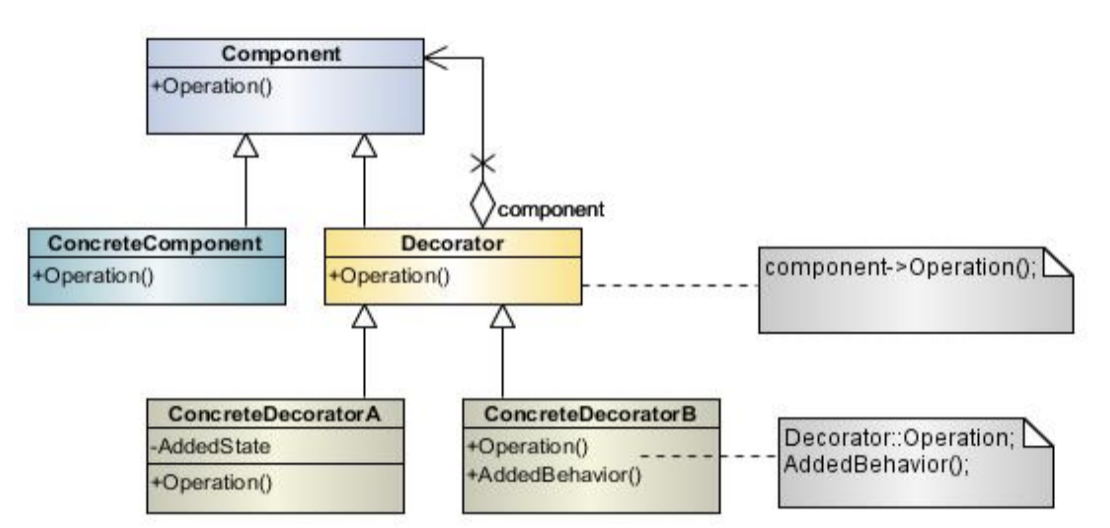
### 2.1.1 Основные характеристики

**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.

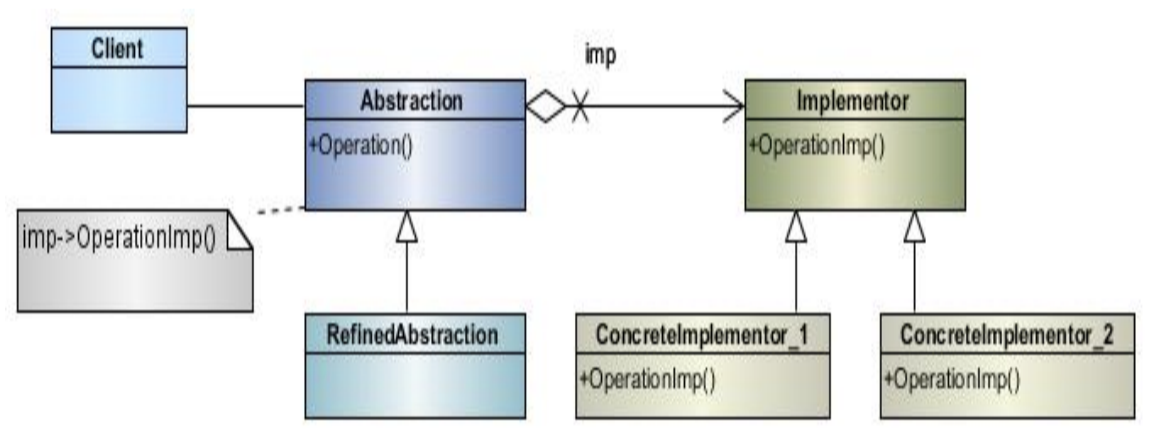
### 2.2.1 Основные характеристики

**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.

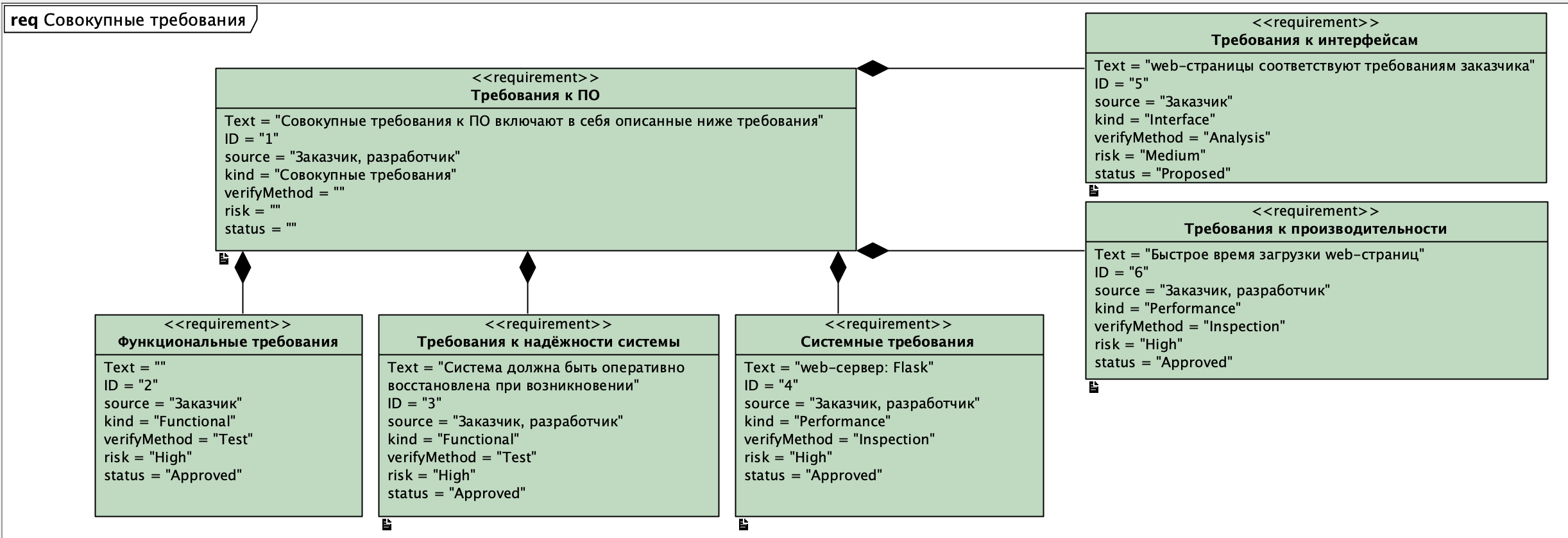
**Реализация**. Паттерн Мост предлагает выделить одну из таких плоскостей в отдельную иерархию классов, храня ссылку на один из её объектов в первоначальном классе.

### 2.2.2. Решение

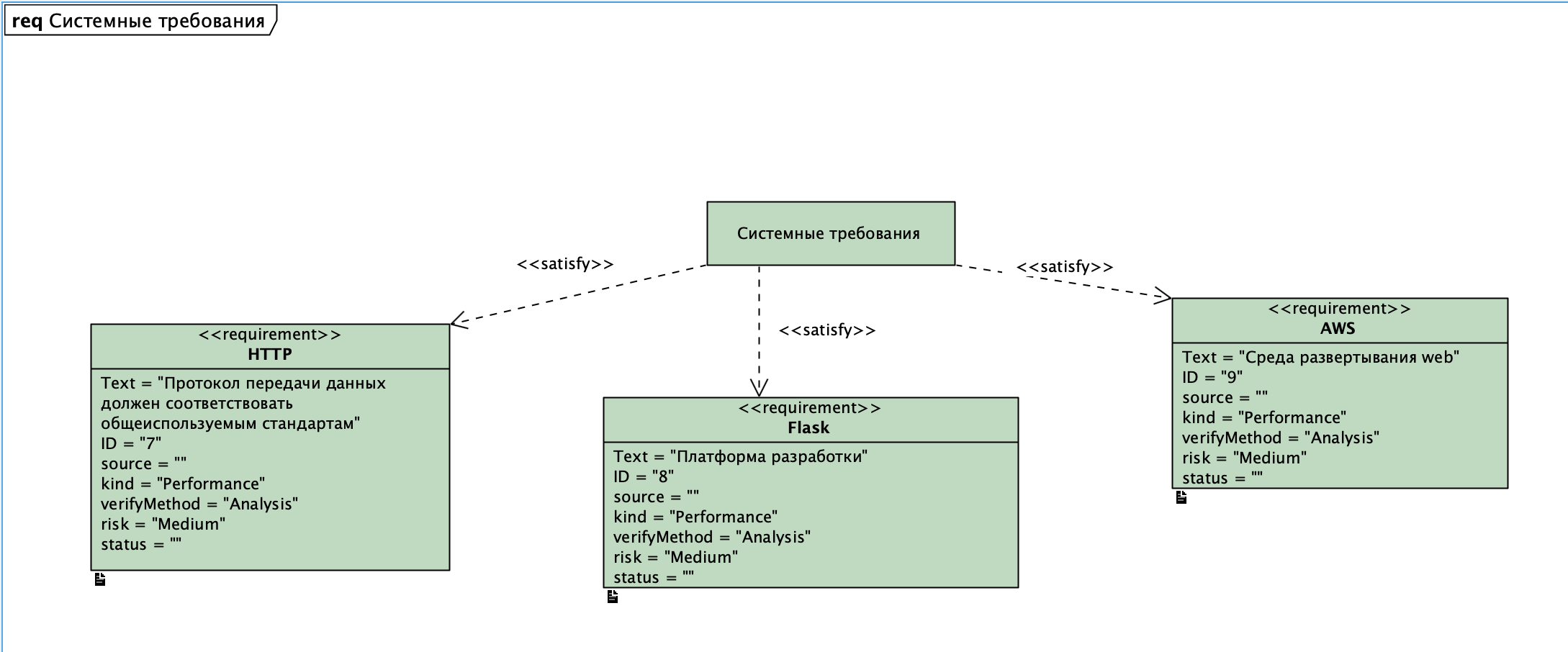


# 3. Спецификация требований

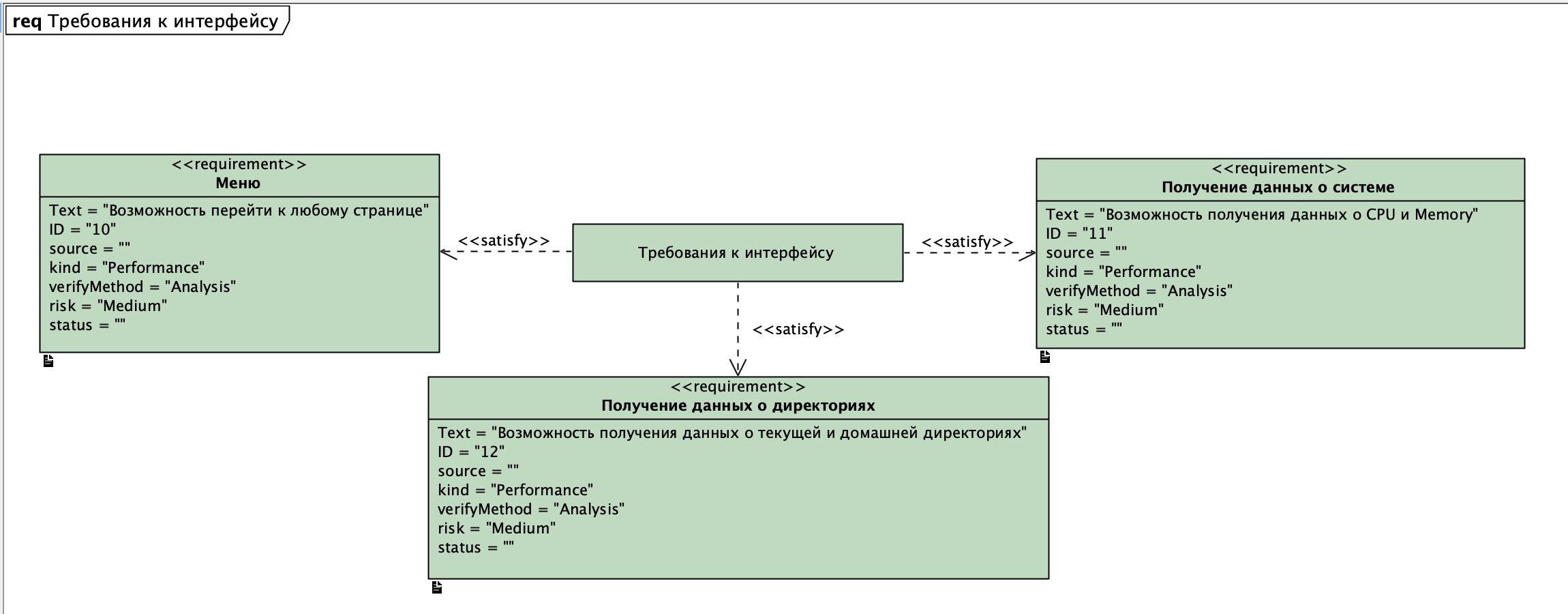
## 3.1 Диаграмма совокупных требований



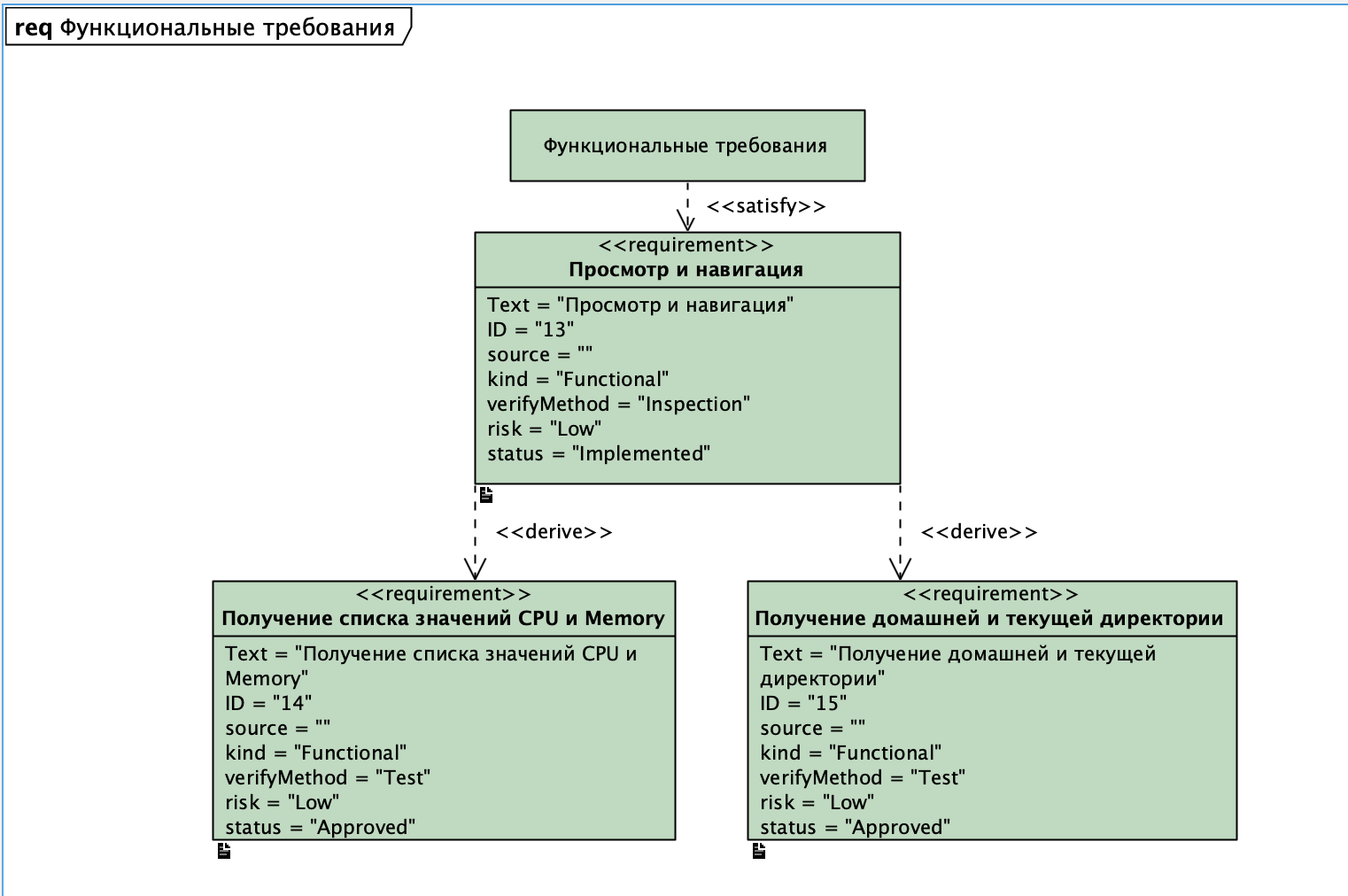
## 3.2 Диаграмма системных требований



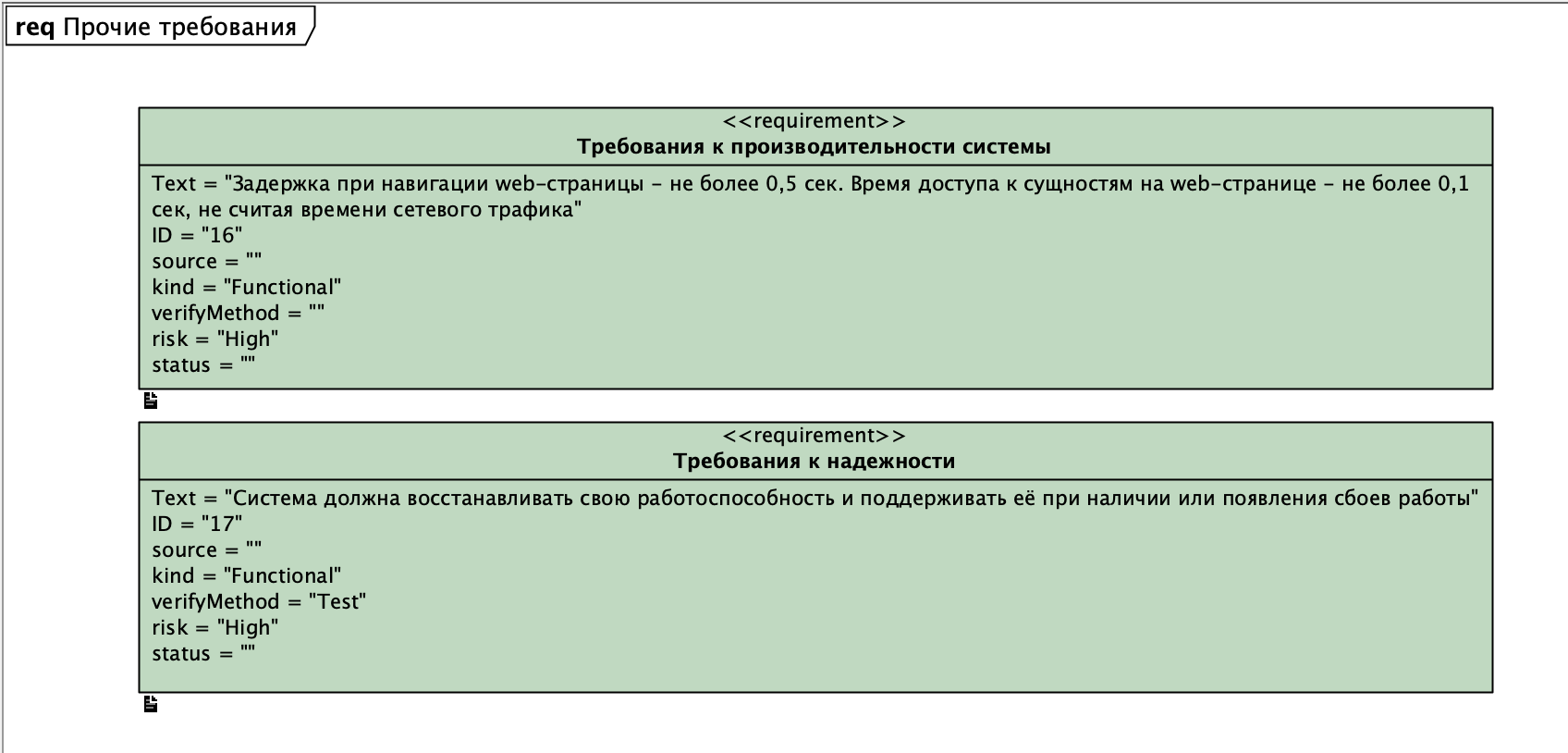
## 3.3 Диаграмма требований к интерфейсу



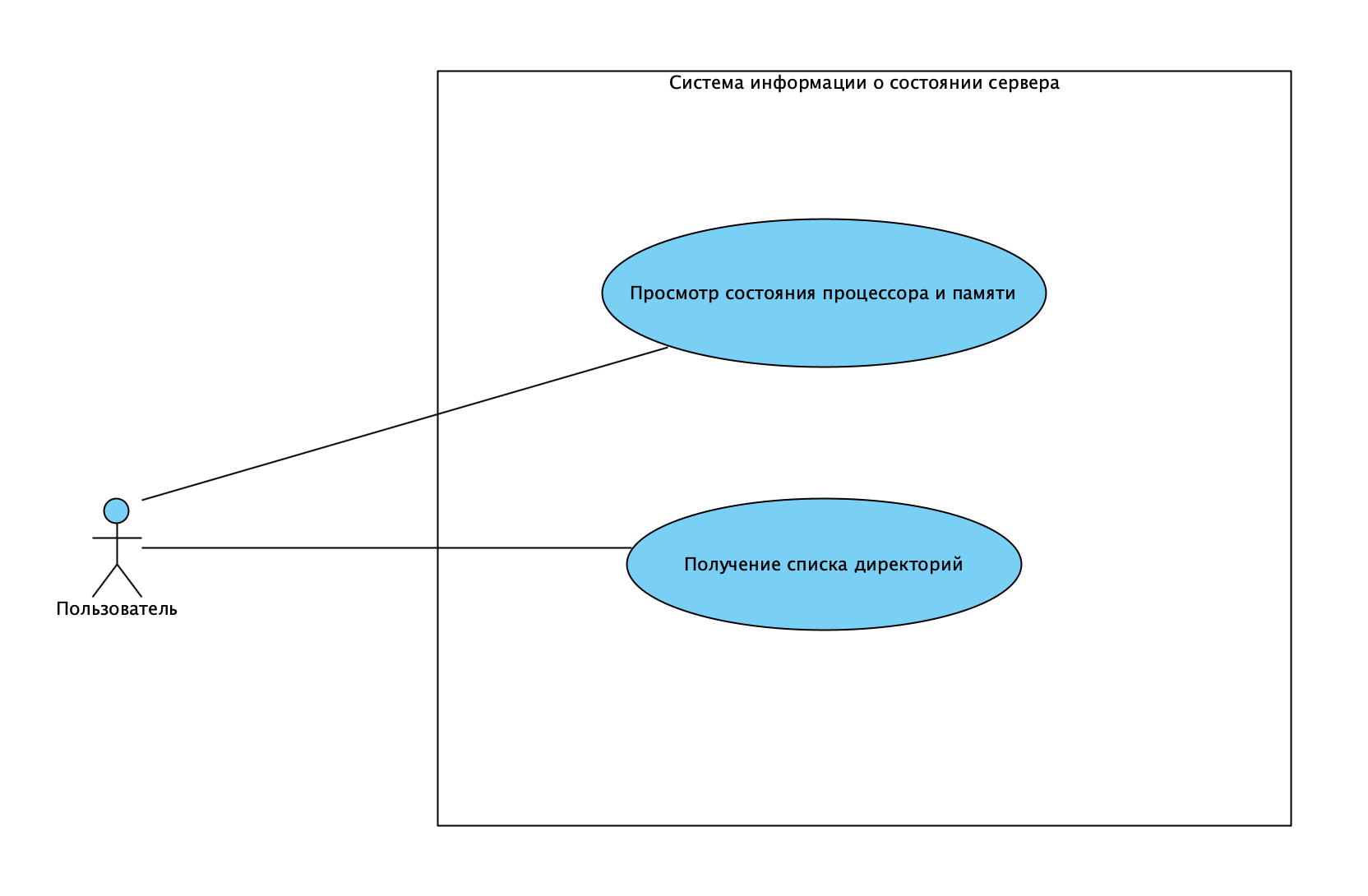
## 3.4 Диаграмма функциональных требований



## 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 платформы, полный путь домашней и текущей директории |

# 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:

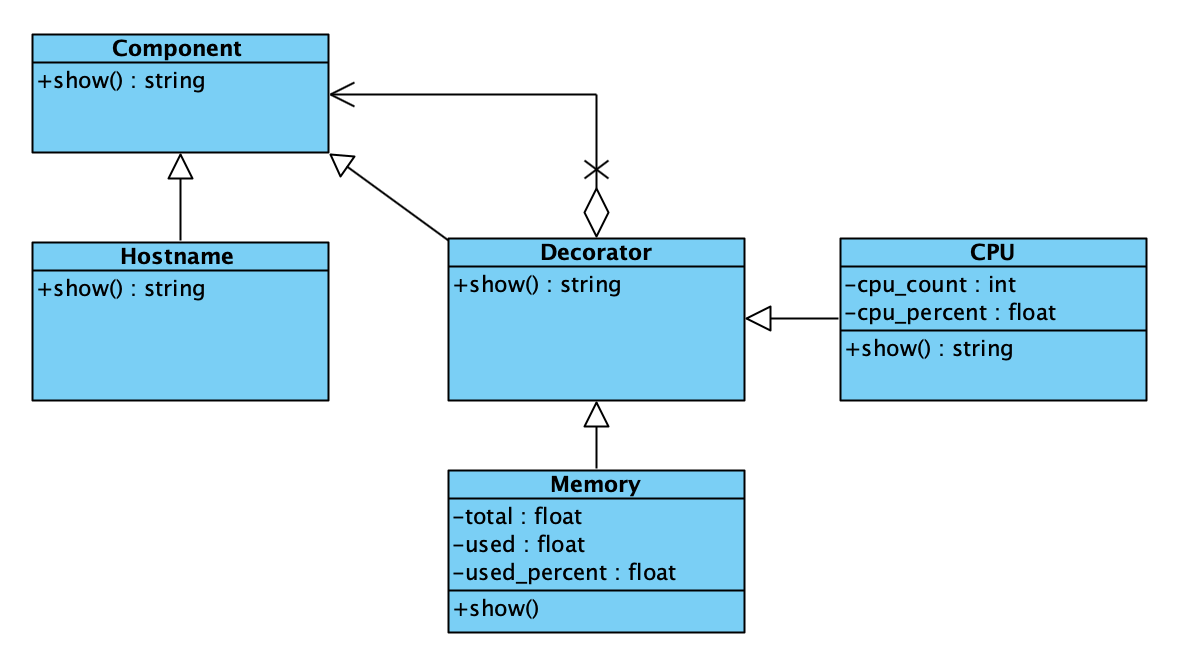
1. **Директория** –– поле private типа “String”. Назначение: полный путь до текущего пользователя сервера.
2. Методы
   1. **операция() –** вернуть текущие значение домашней директории.

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

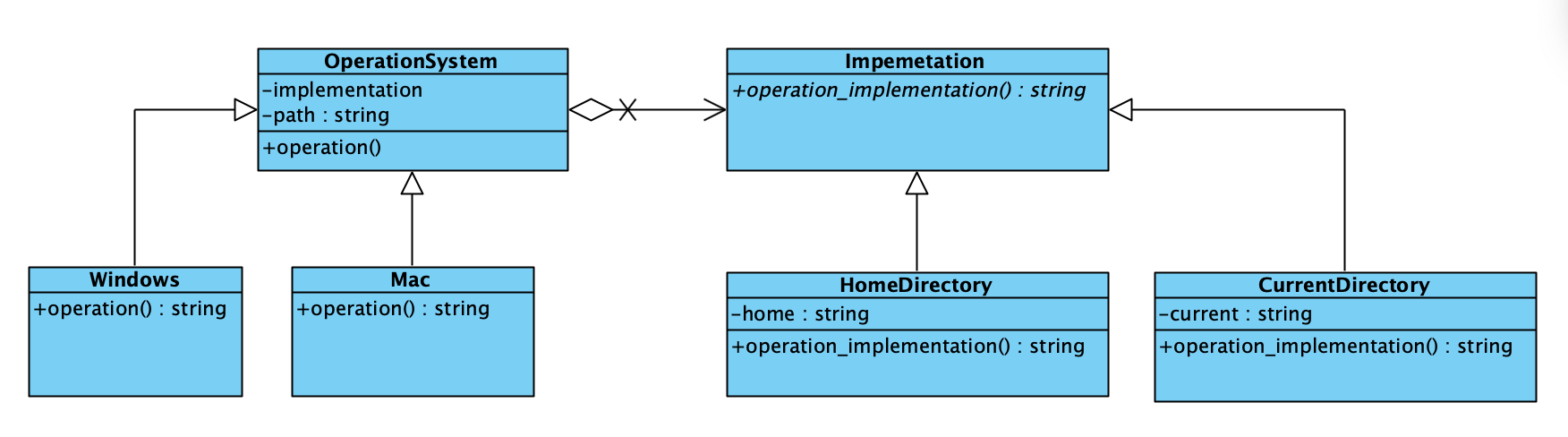
1. **Директория** –– поле private типа “ String”. Назначение: полный путь до исполняемого модуля.
2. Методы
   1. **операция() –** вернуть текущие значение текущей директории.

## 4.1 Диаграмма классов

### 4.1.1 Декоратор

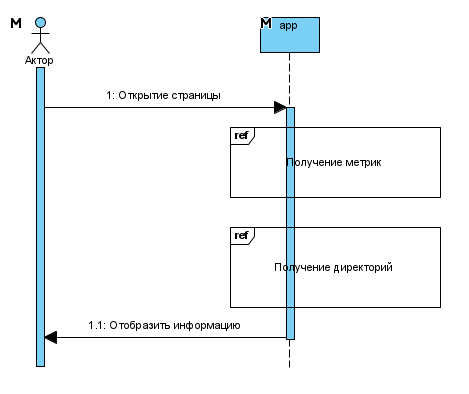


### 4.1.2 Мост

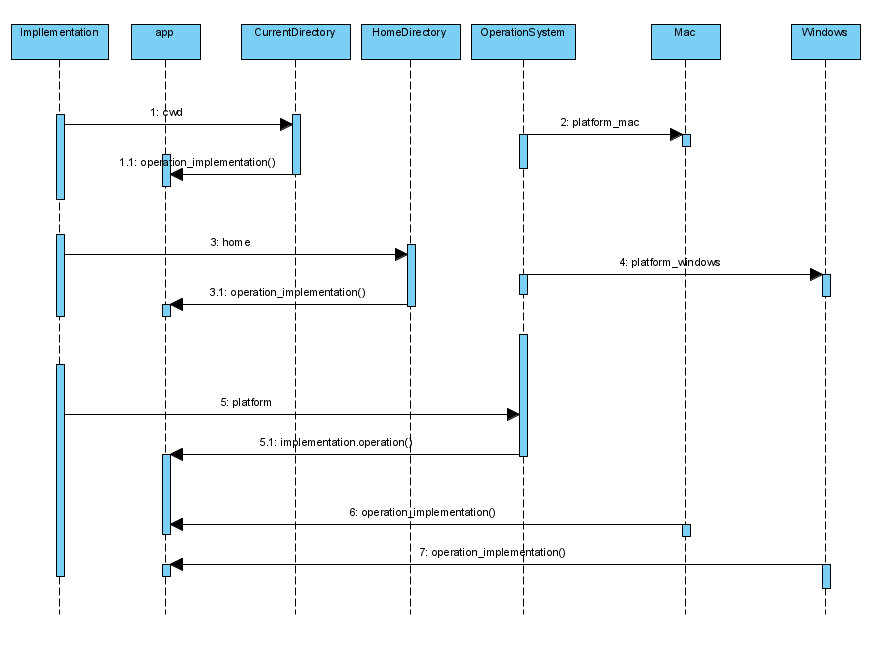


## 4.2 Диаграммы последовательности

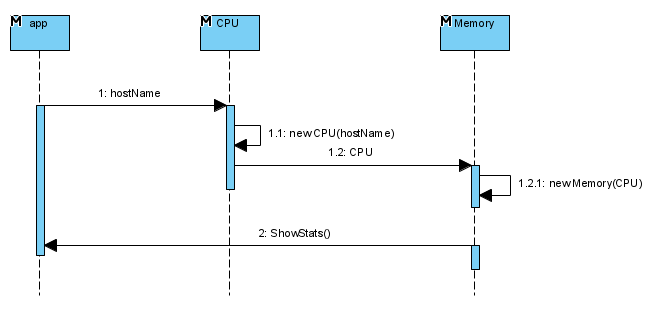
### 4.2.1 Получение информации



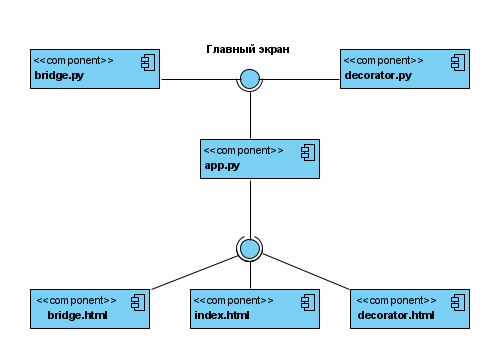
### 4.2.2 Получение директорий



### 4.2.3 Получение метрик

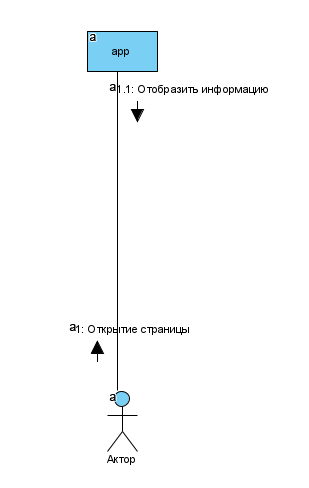


## 4.3 Диаграммы компонентов

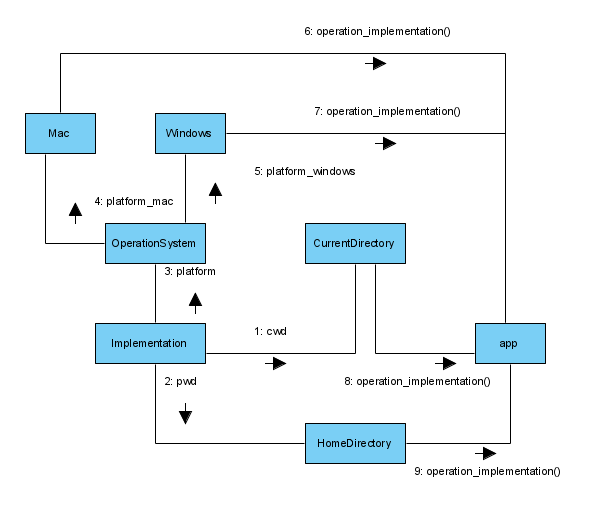


## 4.4 Диаграммы коммуникации

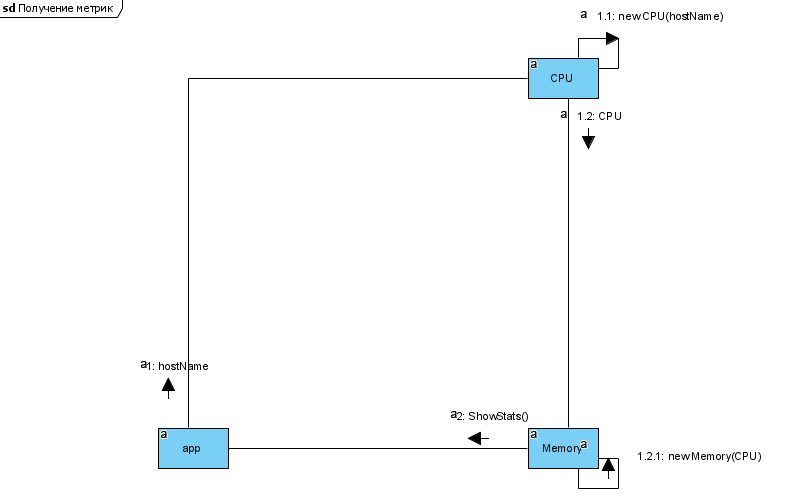
### 4.4.1 Получение информации



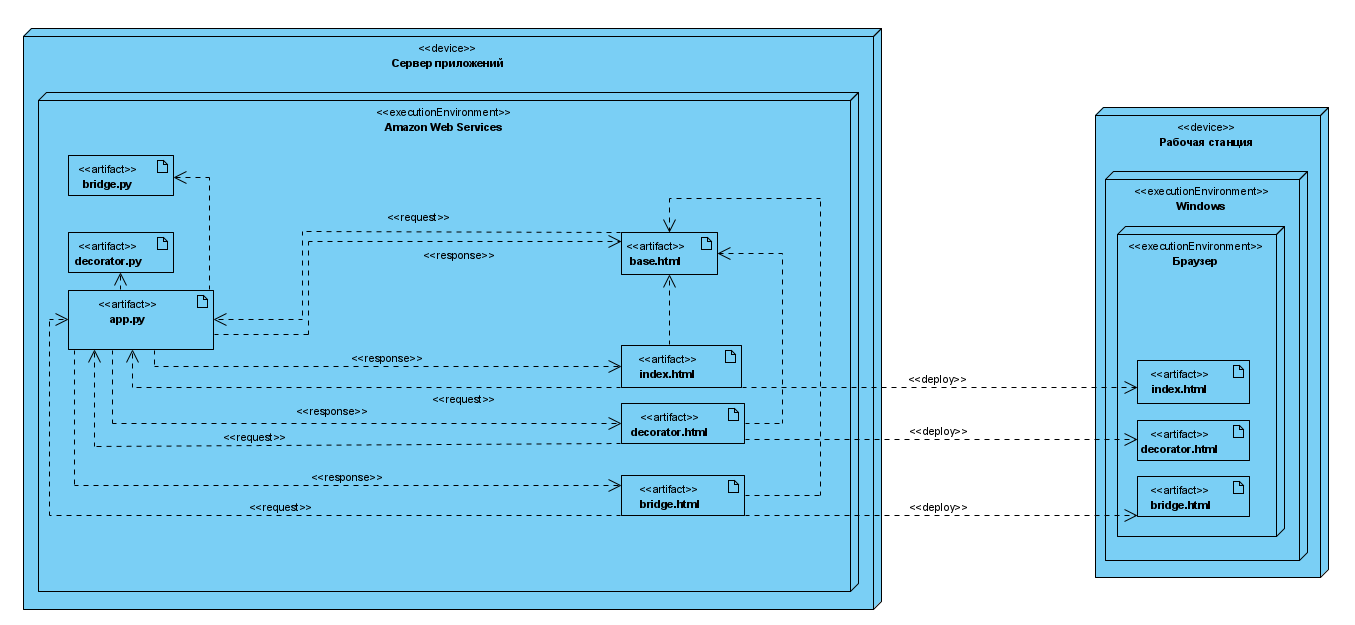
### 4.4.2 Получение директорий



### 4.2.3 Получение метрик



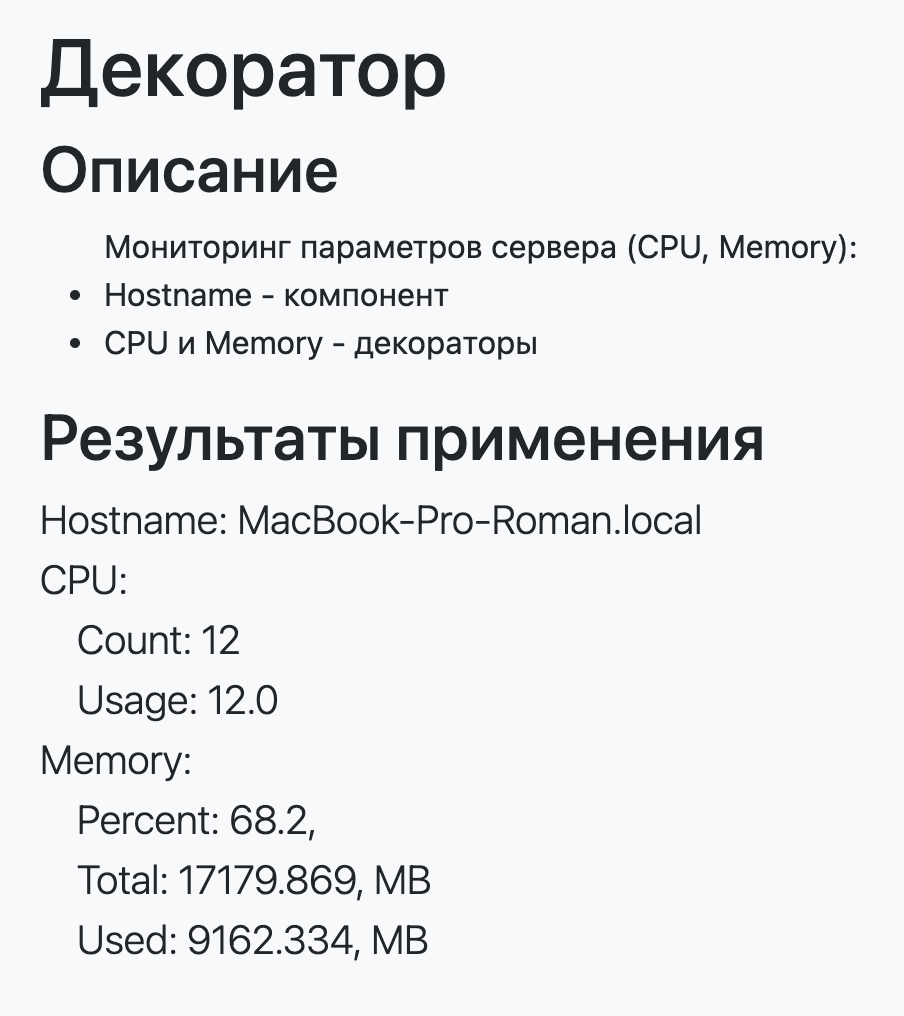
## 4.5 Диаграмма развёртывания



# 5. Пользовательский интерфейс

## 5.1 Пользователь roman

### 5.1.1 Декоратор

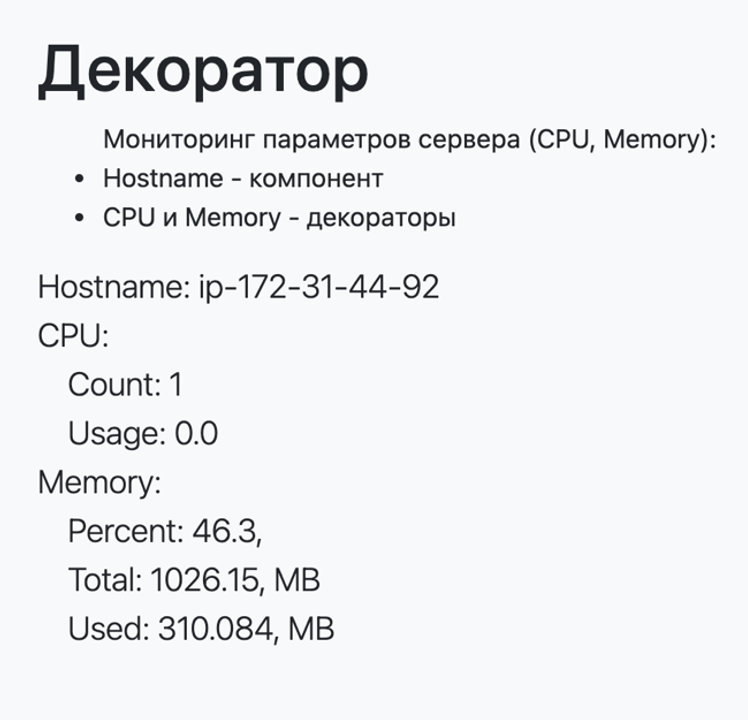


### 5.1.2 Мост

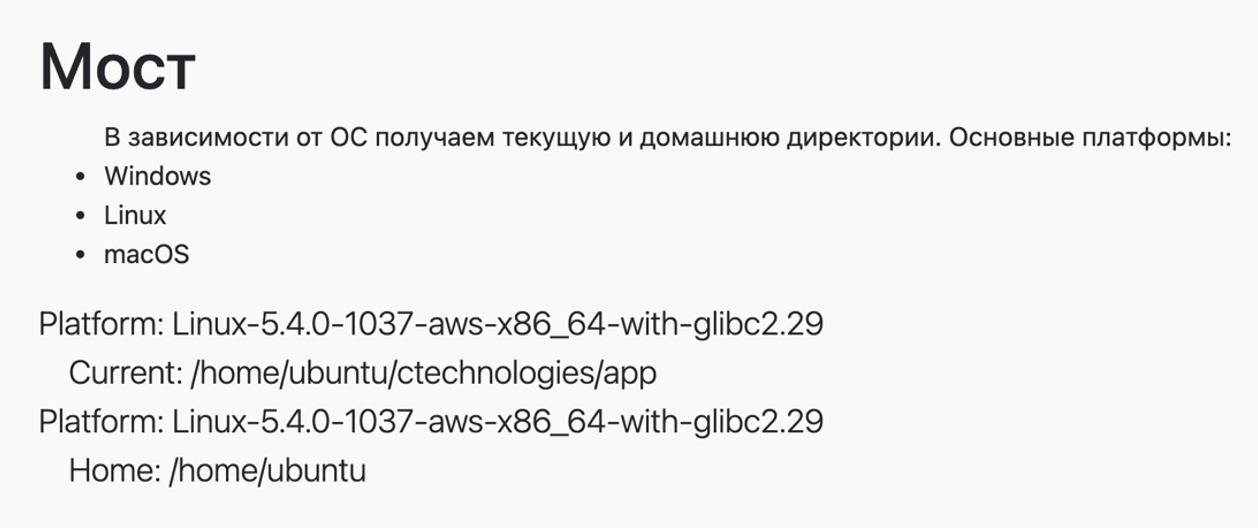


## 5.2 Пользователь ubuntu

### 5.2.1 Декоратор

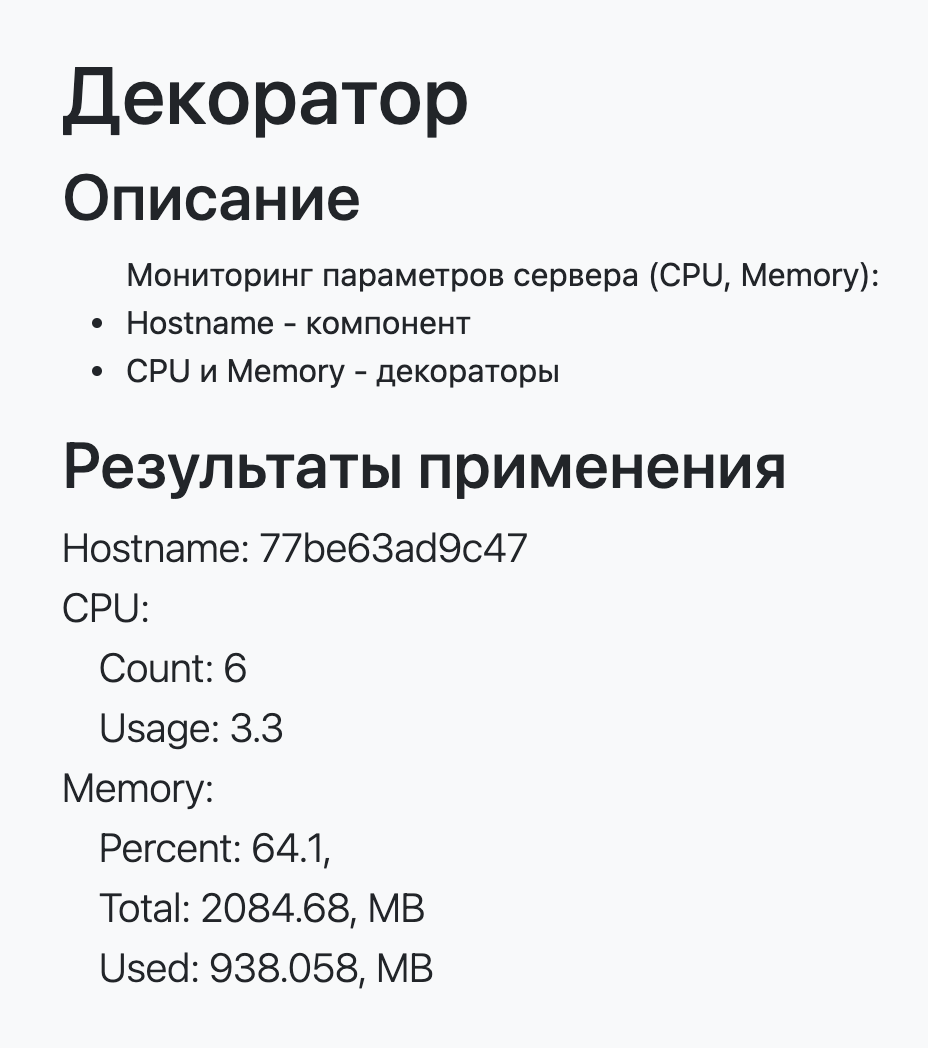


### 5.2.2 Мост

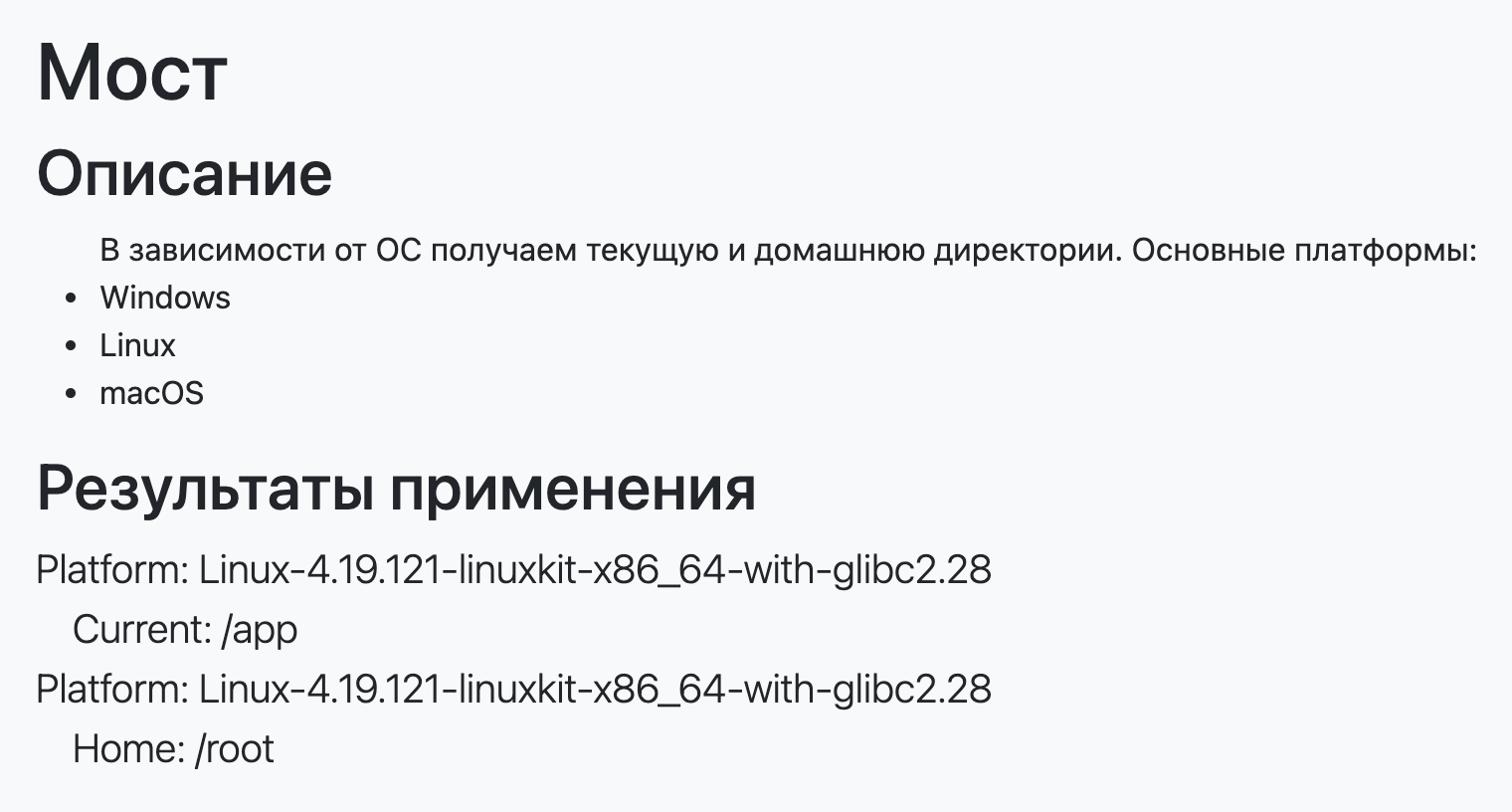


## 5.3 Пользователь root

### 5.3.1 Декоратор



### 5.3.2 Мост



# 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 рефакторинга

### 6.3.1 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.

### 6.3.2 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

Программное обеспечение «Система информации о состоянии сервера» может использовать как:

* Мониторинг состояния процессора и памяти;
* Площадка для обзора текущей директории относительно запускаемого приложения;
* Площадка для обзора домашней директории относительно текущего пользователя;
* Базовый способ мониторинга нагрузки для запускаемого приложения.

## 7.2 Краткое описание возможностей

1. Просмотр состояния загрузки процессора и памяти. Пользователю предоставлена возможность ознакомиться со значениями числа ядер и числом используемых в системе. Для этого ему необходимо на любой странице, перейти по ссылке, располагающейся вверху экрана. Дополнительно пользователь может посмотреть загрузку памяти как в процентах, так и в мегабайтах.
2. Просмотр текущей и домашней директории. Пользователю предоставлена возможность ознакомиться с путём до текущей директории, откуда запущена программа мониторинга. Для этого ему необходимо на любой странице, перейти по ссылке, располагающейся вверху экрана. Дополнительно можно посмотреть путь до домашней директории системного пользователя, который является инициатором запуска данной программы.

## 7.3 Уровень подготовки пользователя

Конечный пользователь программы должен обладать практическими навыками работы в различных браузерах: IE8 и старше (в т.ч. Microsoft Edge), Google Chrome, Mozilla Firefox, Opera, Yandex browser. Обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы. Иметь представление о том, что такое Интернет-банкинг, примерно понимать его структуру.

## 7.4 Назначение программы

### 7.4.1 Виды деятельности, функции

1. Представление имя сервера, где расположена система мониторинга.
2. Представление числа всех ядер процессора.
3. Представление числа используемых ядер процессора.
4. Представление платформы, где расположена система мониторинга.
5. Представление текущей директории, где расположена запускаемая программа.
6. Представление домашней директории пользователя, который запустил данную программу.

### 7.4.2 Программные и аппаратные требования к системе

1. Поддерживаемые операционные системы: Windows XP, Windows 7, Windows 8/8.1, Windows 10;

2. Поддерживаемые браузеры: Google Chrome, IE8 и старше (в т.ч. Microsoft Edge), Opera, Mozilla Firefox, Yandex browser.

### 7.4.3 Подготовка к работе

Для начала работы с «Системой информации о счетах клиентов банка» необходимо открыть браузер (Mozilla Firefox, Google Chrome, IE8 и т.п), для обеспечения лучшей работоспособности рекомендуется использовать браузеры, описанные в п. 7.2 данного руководства.

В адресной строке необходимо ввести имя страницы, на которой представлена «Системой информации о счетах клиентов банка» в зависимости от сервера, на котором будет развернута программа.

Рекомендуемый адрес: http://96.67.22.133:5000

При вводе корректного адреса отобразится главная страница.

## 7.5 Решение известных проблем (Troubleshooting)

В ходе работы с «Система информации о состоянии сервера» могут возникнуть следующие ошибки:

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Ошибка** | **Описание ошибки** | **Требуемые действия пользователя при возникновении ошибки** |
| 1 | Сервер не найден | Возможны проблемы с сетью или с доступом к сайту приложения | Для устранения данной проблемы следует обратиться к администратору |
| 2 | Сайт отобразил 404 ошибка | Код ошибки соответствует спецификации «Not Found». Это означает, что страница не найдена, она перемещена или удалена | Для устранения данной проблемы следует проверить правильность набранного запроса в поисковой строке |
| 3 | Сайт отобразил 500 ошибка | Любая внутренняя ошибка сервера, которая не входит в рамки остальных ошибок класса | Для устранения данной проблемы следует связаться с администрацией сайта |
| 4 | Медленная загрузка сайта | Возможно, в данный момент ведутся технические работы на сайте или модифицируется контент | Для устранения данной проблемы следует воспользоваться сайт через некоторое время |

# 8. Листинг кода

## 8.1 app.py

**import** **platform**

**from** **flask** **import** Flask, render\_template

**from** **bridge** **import** (

OperationSystem,

CurrentDirectory,

HomeDirectory,

Mac

)

**from** **decorator** **import** (

Hostname,

CPU,

Memory

)

app = Flask(\_\_name\_\_)

**def** **html**(text: str) -> str:

**return** text.replace("**\n**", "<br/>").replace("**\t**", "&nbsp;&nbsp;&nbsp;&nbsp;")

**@app**.route('/')

**def** **index**():

**return** render\_template(

'index.html'

)

**@app**.route('/bridge')

**def** **bridge**():

**if** platform.system() == "Darwin":

home: Mac = Mac(CurrentDirectory())

current: Mac = Mac(HomeDirectory())

result: str = f"{home.operation()}**\n**{current.operation()}"

**else**:

home: OperationSystem = OperationSystem(CurrentDirectory())

current: OperationSystem = OperationSystem(HomeDirectory())

result: str = f"{home.operation()}**\n**{current.operation()}"

**return** render\_template(

'bridge.html',

bridge=html(result)

)

**@app**.route('/decorator')

**def** **decorator**():

hostname: Hostname = Hostname()

cpu: CPU = CPU(hostname)

memory: Memory = Memory(cpu)

**return** render\_template(

'decorator.html',

decorator=html(memory.show())

)

## 8.2 bridge.py

**from** **\_\_future\_\_** **import** annotations

**from** **abc** **import** ABC, abstractmethod

**from** **pathlib** **import** Path

**import** **platform**

**class** **OperationSystem**:

"""

Абстракция устанавливает интерфейс для «управляющей» части двух иерархий

классов. Она содержит ссылку на объект из иерархии Реализации и делегирует

ему всю настоящую работу.

"""

**def** **\_\_init\_\_**(self, implementation: Implementation) -> **None**:

self.implementation: Implementation = implementation

**def** **operation**(self) -> str:

**return** f"Platform: {platform.platform()}" \

f"**\n\t**{self.implementation.operation\_implementation()}"

**class** **Mac**(OperationSystem):

"""

Можно расширить Абстракцию без изменения классов Реализации.

"""

**def** **operation**(self) -> str:

**return** f"Platform: {platform.platform()}" \

f"**\n\t**{self.implementation.operation\_implementation()}"

**class** **Windows**(OperationSystem):

"""

Можно расширить Абстракцию без изменения классов Реализации.

"""

**def** **operation**(self) -> str:

**return** f"Platform: Windows**\n\t**{self.implementation.operation\_implementation()}"

**class** **Implementation**(ABC):

"""

Реализация устанавливает интерфейс для всех классов реализации. Он не должен

соответствовать интерфейсу Абстракции. На практике оба интерфейса могут быть

совершенно разными. Как правило, интерфейс Реализации предоставляет только

примитивные операции, в то время как Абстракция определяет операции более

высокого уровня, основанные на этих примитивах.

"""

**@abstractmethod**

**def** **operation\_implementation**(self) -> str:

**pass**

"""

Каждая Конкретная Реализация соответствует определённой платформе и реализует

интерфейс Реализации с использованием API этой платформы.

"""

**class** **CurrentDirectory**(Implementation):

**def** **operation\_implementation**(self) -> str:

**return** f"Current: {Path.cwd()}"

**class** **HomeDirectory**(Implementation):

**def** **operation\_implementation**(self) -> str:

**return** f"Home: {Path.home()}"

**def** **client\_code**(abstraction: OperationSystem) -> str:

"""

За исключением этапа инициализации, когда объект Абстракции связывается с

определённым объектом Реализации, клиентский код должен зависеть только от

класса Абстракции. Таким образом, клиентский код может поддерживать любую

комбинацию абстракции и реализации.

"""

**return** abstraction.operation()

**if** \_\_name\_\_ == "\_\_main\_\_":

"""

Клиентский код должен работать с любой предварительно сконфигурированной

комбинацией абстракции и реализации.

"""

implementation: CurrentDirectory = CurrentDirectory()

abstraction: OperationSystem = OperationSystem(implementation)

print(client\_code(abstraction))

implementation: HomeDirectory = HomeDirectory()

abstraction: Windows = Windows(implementation)

print(client\_code(abstraction))

## 8.3 decorator.py

**from** **socket** **import** gethostname

**import** **psutil**

**class** **Component**:

"""

Базовый интерфейс Компонента определяет поведение, которое изменяется

декораторами.

"""

**def** **show**(self) -> str:

**pass**

**class** **Hostname**(Component):

"""

Конкретные Компоненты предоставляют реализации поведения по умолчанию. Может

быть несколько вариаций этих классов.

"""

**def** **show**(self) -> str:

**return** f"Hostname: {gethostname()}"

**class** **Decorator**(Component):

"""

Базовый класс Декоратора следует тому же интерфейсу, что и другие

компоненты. Основная цель этого класса - определить интерфейс обёртки для

всех конкретных декораторов. Реализация кода обёртки по умолчанию может

включать в себя поле для хранения завёрнутого компонента и средства его

инициализации.

"""

\_component: Component = **None**

**def** **\_\_init\_\_**(self, component: Component) -> **None**:

self.\_component = component

**@property**

**def** **component**(self) -> Component:

"""

Декоратор делегирует всю работу обёрнутому компоненту.

"""

**return** self.\_component

**def** **show**(self) -> str:

**return** self.\_component.show()

**class** **CPU**(Decorator):

"""

Конкретные Декораторы вызывают обёрнутый объект и изменяют его результат

некоторым образом.

"""

**def** **show**(self) -> str:

"""

Декораторы могут вызывать родительскую реализацию операции, вместо того,

чтобы вызвать обёрнутый объект напрямую. Такой подход упрощает

расширение классов декораторов.

"""

**return** f"{self.component.show()}" \

f"**\n**CPU:" \

f"**\n\t**Count: {psutil.cpu\_count()}" \

f"**\n\t**Usage: {psutil.cpu\_percent(interval=1)}"

**class** **Memory**(Decorator):

"""

Декораторы могут выполнять своё поведение до или после вызова обёрнутого

объекта.

"""

**def** **show**(self) -> str:

stats = psutil.virtual\_memory()

total = stats.total

used = stats.used

used\_percent = stats.percent

**return** f"{self.component.show()}" \

f"**\n** Memory:" \

f"**\n\t**Percent: {used\_percent}," \

f"**\n\t**Total: {round(total / 1e+6, 3)}, MB" \

f"**\n\t**Used: {round(used / 1e+6, 3)}, MB"

**def** **client\_code**(component: Component) -> str:

"""

Клиентский код работает со всеми объектами, используя интерфейс Компонента.

Таким образом, он остаётся независимым от конкретных классов компонентов, с

которыми работает.

"""

**return** component.show()

**if** \_\_name\_\_ == "\_\_main\_\_":

hostname: Hostname = Hostname()

cpu: CPU = CPU(hostname)

memory: Memory = Memory(cpu)

decorator: str = client\_code(memory)

print(decorator)

## 8.4 base.html

<!doctype html>

<html lang="ru">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"

integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">

<title>{% **block** title %} {% **endblock** %}</title>

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container-fluid">

<a class="navbar-brand" href="{{ url\_for('index') }}">Облачные сервисы и технологии</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav"

aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item">

<a class="nav-link" href="{{ url\_for('decorator') }}">Декоратор</a>

</li>

<li class="nav-item">

<a class="nav-link" href="{{ url\_for('bridge') }}">Мост</a>

</li>

</ul>

</div>

</div>

</nav>

<div class="container">

{% **block** content %} {% **endblock** %}

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"

integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"

crossorigin="anonymous"></script>

</body>

</html>

## 8.5 index.html

{% **extends** 'base.html' %}

{% **block** title %} Облачные технологии и сервисы {% **endblock** %}

{% **block** content %}

<div class="bg-light p-5 rounded">

<h1> Проектное задание</h1>

<br/>

<h2> Облачные сервисы и технологии</h2>

<br/>

<h3> Паттерны: Декоратор. Мост</h3>

<br/>

<p>Группа: ИВТ-М20<br/>Cтудент: Лискунов Р. Г.</p>

</div>

{% **endblock** %}

## 8.6 bridge.html

{% **extends** 'base.html' %}

{% **block** title %} Мост {% **endblock** %}

{% **block** content %}

<div class="bg-light p-5 rounded">

<h1>Мост</h1>

<h2>Описание</h2>

<ul>В зависимости от ОС получаем текущую и домашнюю директории. Основные платформы:

<li>Windows</li>

<li>Linux</li>

<li>macOS</li>

</ul>

<h2>Результаты применения</h2>

<p class="lead">{{ bridge | **safe** }}</p>

</div>

{% **endblock** %}

## 8.7 decorator.html

{% **extends** 'base.html' %}

{% **block** title %} Декоратор {% **endblock** %}

{% **block** content %}

<div class="bg-light p-5 rounded">

<h1>Декоратор</h1>

<h2>Описание</h2>

<ul>

Мониторинг параметров сервера (CPU, Memory):

<li>Hostname - компонент</li>

<li>CPU и Memory - декораторы</li>

</ul>

<h2>Результаты применения</h2>

<p class="lead">{{ decorator | **safe** }} </p>

</div>

{% **endblock** %}

# 9. Протокол изменений, внесенный в первоначальный проект

9.1 В соответствии с готовым продуктом в диаграмму компонентов, диаграмму классов и диаграмму развёртывания были внесены изменения:

9.1.1. Была пересмотрена структура диаграммы классов.

9.1.2. Была пересмотрена связь компонентов и передача параметров.

9.1.3. Были внесены корректировки в диаграмму компонентов.

9.1.4. Диаграмма развёртывания стала более понятной и информативной.

9.2 Улучшен интерфейс системы информирования о состоянии сервера.

9.3. Внесены мелкие исправления в другие классы.