Удалённый мониторинг ПК

# Цели:

1. Удалённо просматривать загруженность метрик систем.
2. Получения уведомлений об ошибках.

# Вступление:

При работе с сервером или кластером серверов, требуется непрерывная работа каждого из них. Выход из строя хотя бы одного сервера, может существенно повлиять на работу всей системы. Сбой может произойти по разным причинам: недостаточно места на жёстком диске, ненужный процесс который потребляет много ресурсов, таких причин бесчисленное множество. Что бы решить данную проблему необходим сервис, который будет производить мониторинг системы круглосуточно и сообщать о превышении значения какой-либо метрики снятой с определённого сервера (ПК).

# Используемые технологии:

ПО реализовано на языке программирования Java с использованием Spring MVC, Hibernate Framework и highcharts. Для работы ПО необходима БД (в примере используется postgresql).

# Краткое писание:

System Monitoring (SM) – это приложение, позволяющее непрерывно отслеживать состояние хоста, снимать значения его метрик, предоставлять это все в удобном для пользователя виде и оповещать пользователя о проблемах связанных с доступностью хостов или со значениями метрик. UI (Пользовательский интерфейс) реализован в виде веб сайта, это удобно тем, что пользователь в любой момент из любого места (где есть выход в интернет) сможет использовать данный сервис.

В SM, функции доступные пользователю, изначально доступны в зависимости от одной из двух ролей: пользователь или администратор.

Возможности, предоставленные пользователю:

• Просмотр сведений о хосте

• Просмотр возникших проблем

• Система оповещения

• Графическое представление для состояния метрик

• Добавление графиков на главную страницу

Администратору предоставлены все возможности пользователя, а также следующие функции:

• Редактирование хостов

• Редактирование локальных метрик

• Редактирование шаблонных метрик

• Редактирование аккаунтов

# Архитектура SM:

Архитектуру можно разбить на несколько частей (Рис.1):

1. UI – интерфейс пользователя.
2. Controllers – слой который обеспечивает связь между пользователем и системой.
3. Agents – Агенты отвечающие за определённые сервисы.
4. DB – Слой базы данных, для хранения необходимой информации.

E:\downloads\Untitled Diagram (1).png

Рис.1 “Архитектура приложения”

Доступ к базе данных разделён на два способа:

1. Через реализацию IStorageController.
2. Через интерфейс JPA.

Сервис “SchedulerTask” запускает по расписанию Агентов, которые отвечают за подключение к хосту, считывание значений метрик с хостов и за отправку уведомлений.

Уведомления в системе делятся на два типа:

1. Отправка сообщений на почту.
2. Отправка уведомления на UI пользователя.

Контроллеры обеспечивают связь между пользователем и системой: контролируют ввод данных пользователем и используют модель и представление для реализации необходимой реакции.

Слой UI представлен в виде Web интерфейса, который использует технологию Thymeleaf для построения страниц.

# Детальный дизайн ядра системы:

Центральной частью ПО является Spring Framework, который отвечает за управление жизненным циклом объектов: создание объектов, вызов методов инициализации и конфигурирование объектов путем связывания их между собой. Конфигурирование контейнера осуществляется путем аннотаций и класса DatabaseConfig, содержащих информацию, необходимую для создания bean’ов.

После запуска приложения Spring разворачивает Web сервер и инициализирует beans, далее с помощью аннотации Scheduler, каждые 10 секунд сервис “ScheduledTask” запускает считывание метрик с удалённых хостов. Диаграмма классов для сервиса “ScheduledTask” представлена на Рис.2

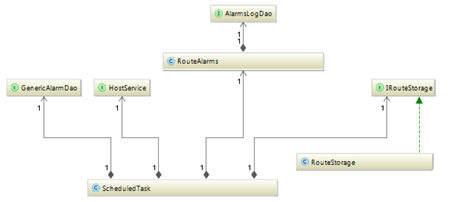


Рис.2 “Диаграмма классов ScheduledTask”

Сервис “ScheduledTask” поочерёдно считывает информацию о каждом хосте, которая хранится в базе, создаёт объект класса SSHAgent, который по сетевому протоколу SSH подключается к удалённому хосту.

Если подключение не удалось установить, то с помощью сервиса “RouteAlarms” происходит отправка сообщения об ошибке (Рис.3).

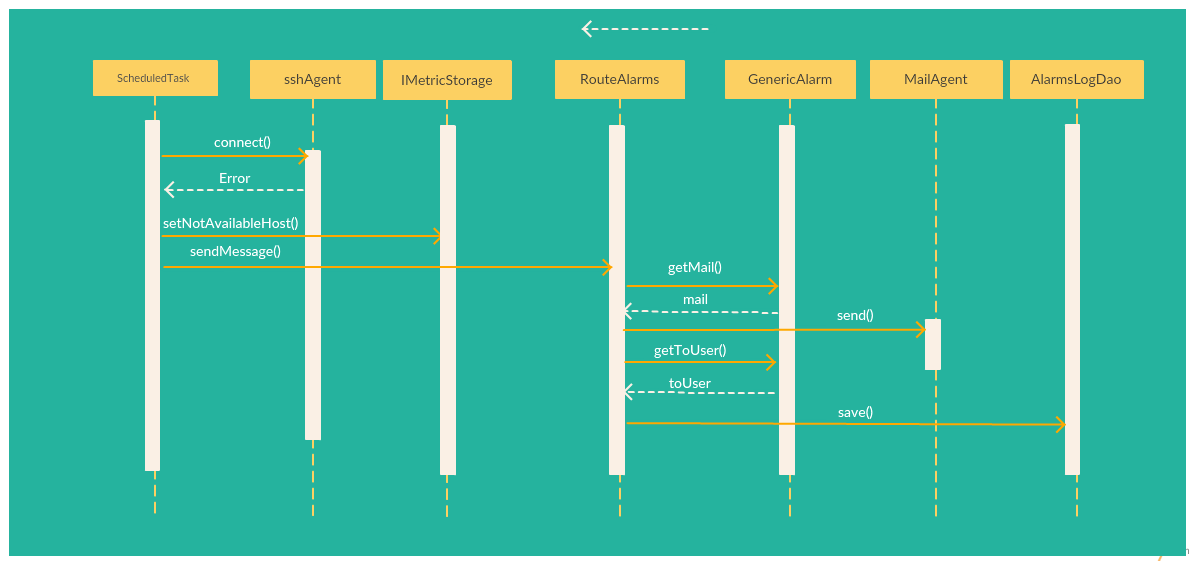


Рис.3 “Оповещение пользователя о неудачной попытке подключения”

После успешного подключения IStorageController возвращает List<InstanceMetric> для данного хоста. С каждой метрики SSHAgent считывает и передаёт команду в созданный канал. После выполнения команды, хост возвращает результат её выполнения. После обработки результата, IStorageController заносит его в базу данных (Рис.4).

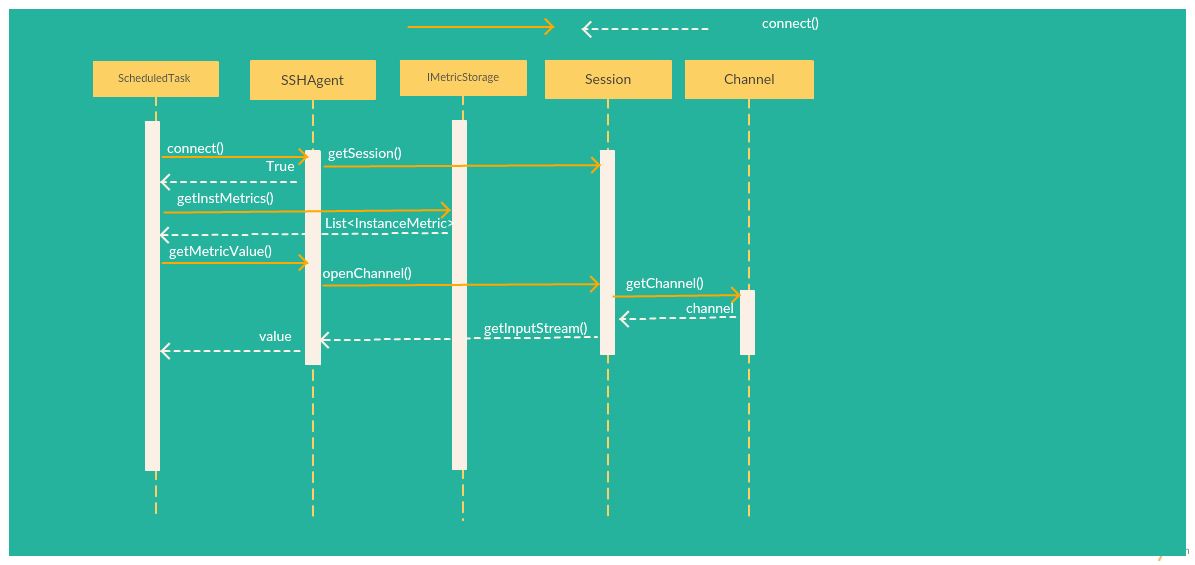


Рис.4 “Снятие значений метрик с хоста”

Сервис “ScheduledTask” сравнивается полученное значения с пороговыми записанными в InstanceMetric. При необходимости с помощью сервиса “RouteAlarms” происходит отправка сообщения об ошибке.

Для работы с БД Spring Framework создаёт, хранит и инжектит в необходимые нам классы сервис “IStorageController”, отвечающий за логику работы с БД. Все действия пользователя через Web интерфейс обрабатываются определённым контроллером, который в свою очередь использует сервис “ IStorageController ” (Рис. 5).

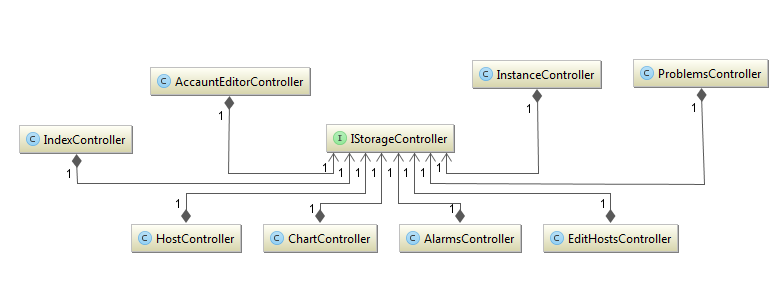


Рис.5 “Диаграмма классов IStorageController”

Например, пользователь заходит на страницу со всеми проблемами (Рис.6).

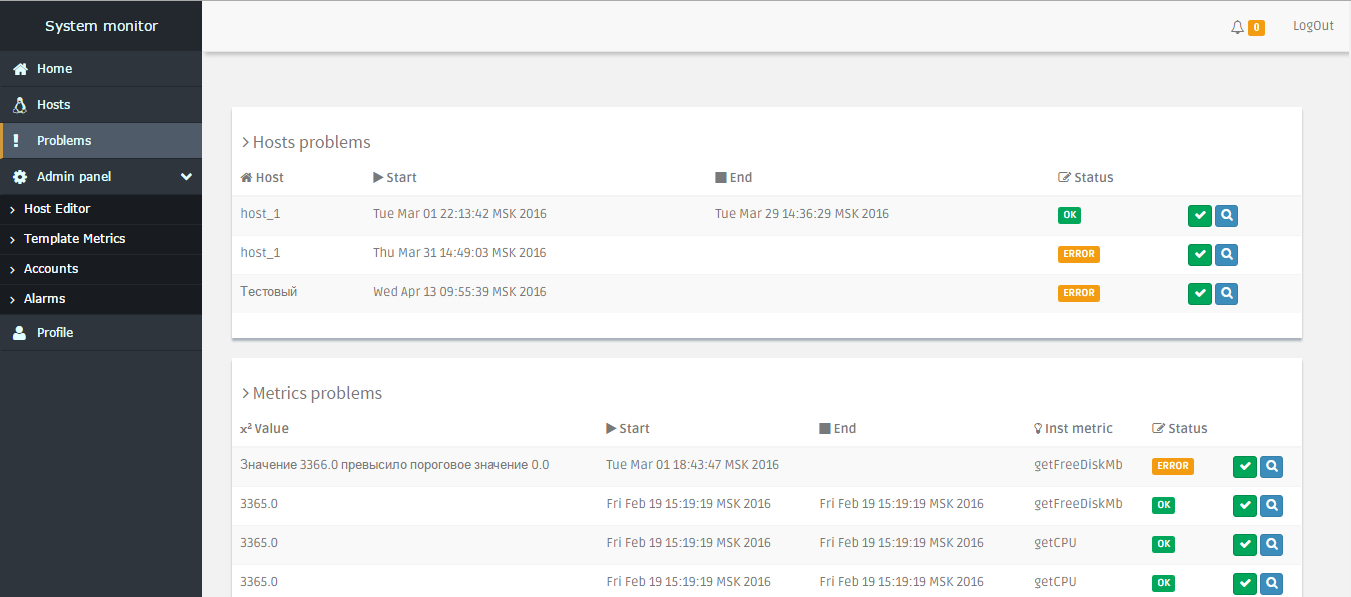


Рис.6 “Информация о проблемах”

Для построения данной страницы выполняется некоторая последовательность действий:

1. Spring предоставляет в контроллер, отвечающий за данную страницу, необходимую реализацию интерфейса IStorageController.
2. Контроллер вызывает метод IStorageController.getMetricProblems().
3. Данный метод обращается в базу через метод queryForList(String) объекта JdbcTemplate. Результатом запроса будет список List<Map<String, Object>>.
4. Проходя по списку создаётся List<MetricState>.
5. Реализация IStorageController возвращает в контроллер список проблем с метриками List<MetricState>.
6. Данный список помещается в ModelAndView.
7. ModelAndView передаётся в представление и обрабатывается с помощью thymeleaf.
8. После обработки пользователю возвращается HTML страница.

Для проблем связанных с хостами аналогичная последовательность.

В приложении реализовано графическое представление для снятых значений метрик (Рис.7).

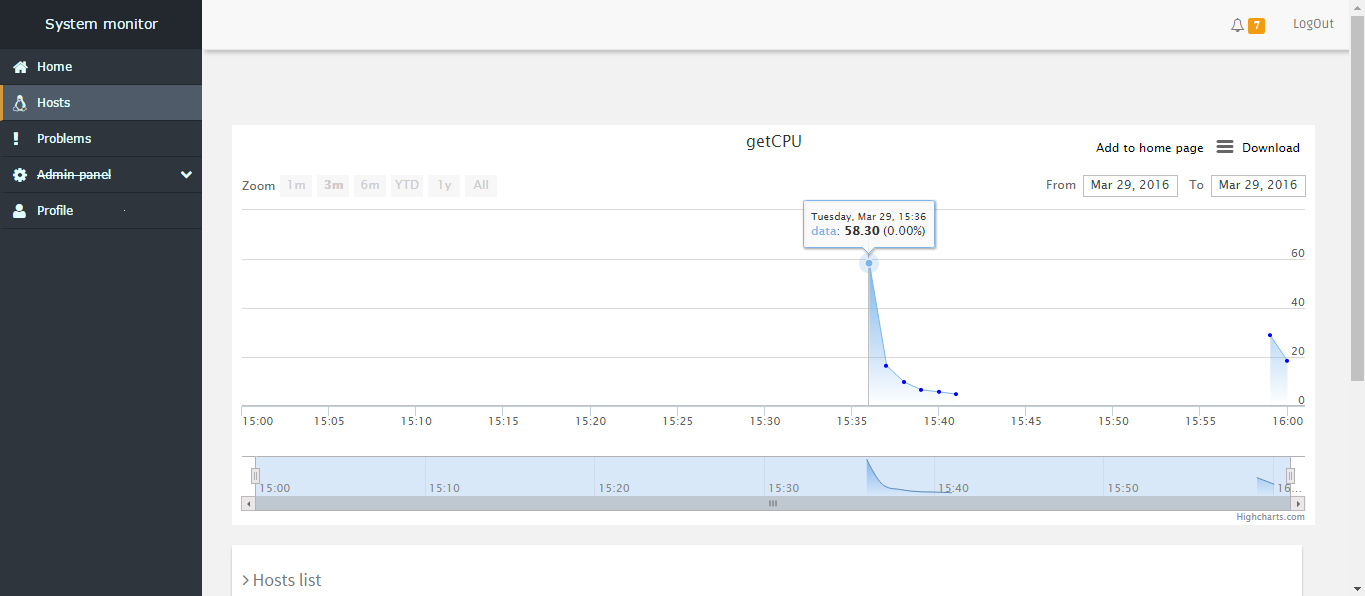


Рис.7 “Графическое представление снятых значений”

Данная функция реализована с помощью Highcharts. Highcharts — библиотека для создания чартов написанная на языке JavaScript [1].

Для построения графика выполняется некоторая последовательность действий (Рис.8):

1. Выполняется ajax запрос, который передаёт id метрики и дату (если дата не указана, то построение графика будет производиться от последнего снятого значения) в контроллер.
2. Контроллер обращается в IChartStorage и вызывает необходимый метод.
3. Реализация IChartStorage в свою очередь посылает запрос в базу данных и принимает последовательность значений метрики за определённый период времени.
4. Все полученные значения проходят через механизм агрегации, где формируется объект класса GraphPoints
5. После агрегации объект GraphPoints возвращается через контроллер в javascript.
6. Javascript на основе данных значений производит построение графика и предоставляет его пользователю.

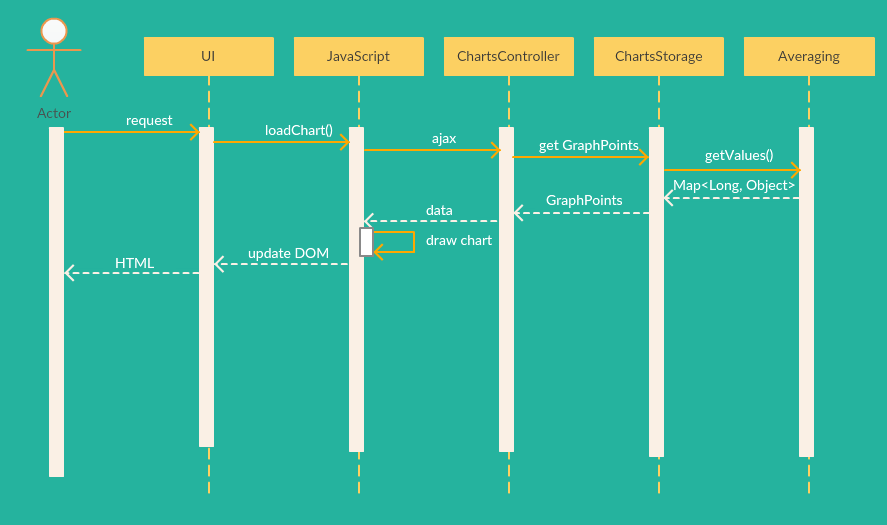


Рис.8 “Отрисовка графика”

Так, как количество ‘точек’ за некоторый период может быть очень большим, то необходима агрегация данных. Агрегация производится для “масштаба” час, день, неделя, месяц и год. Для агрегации высчитывается среднее арифметическое за предыдущий “уровень” масштаба и производится запись этого значения и времени в объект Map<Long, Object>. Если не производился съём значений метрики на некотором промежутке времени, то вместо значения записывается null, это необходимо для обрыва графика в данном месте.

При необходимости пользователь может поменять масштаб графика. Для этого необходимо:

1. Навести курсор мыши на нужный график.
2. Прокруткой колёсика поменять масштаб.

При прокрутке колёсика javascript считывает атрибут “zoomCount”, который записан в div’e, куда был помещён график. В зависимости от значения данного атрибута вызывается метод контроллера, который отвечает за определённый масштаб графика. После успешного выполнения атрибут “zoomCount” перезаписывается и обновляются все точки графика.

Данные в базе логически разделены на две части, с одной из которых взаимодействует реализация интерфейса IStorageController, с другой доступ реализован через hibernate.

Таблицы с которыми взаимодействует реализация интерфейса IStorageController:

1. VALUE\_METRIC – Хранит снятые значения метрик с хостов.
2. TEMPLATE\_METRICS – Шаблонные метрики, метрики которые часто используются.
3. INSTANCE\_METRIC – Метрики которые привязаны к определённому хосту.
4. METRIC\_STATE – Таблица хранит информацию о состоянии instance метрик.
5. HOST\_STATE – Таблица хранит информацию о состоянии хостов.
6. FAVORITES – Таблица хранит информацию о тех графиках, которые хочет видеть пользователь на главной странице приложения.
7. Users – Таблица с пользователями.
8. Roles – Таблица с ролями пользователей.

Взаимодействие с помощью hibernate используется для таблиц:

1. sshconfigurationhibernate – Таблица содержит информацию о хостах.
2. genericalarm – Таблица содержит информацию о том, какой текст сообщения и куда его оправить.
3. log\_alarms – Таблица нужна для вывода и хранении сообщений, которые были отправлены на интерфейс пользователя.

ER-model базы данных представлена на рисунке 9.

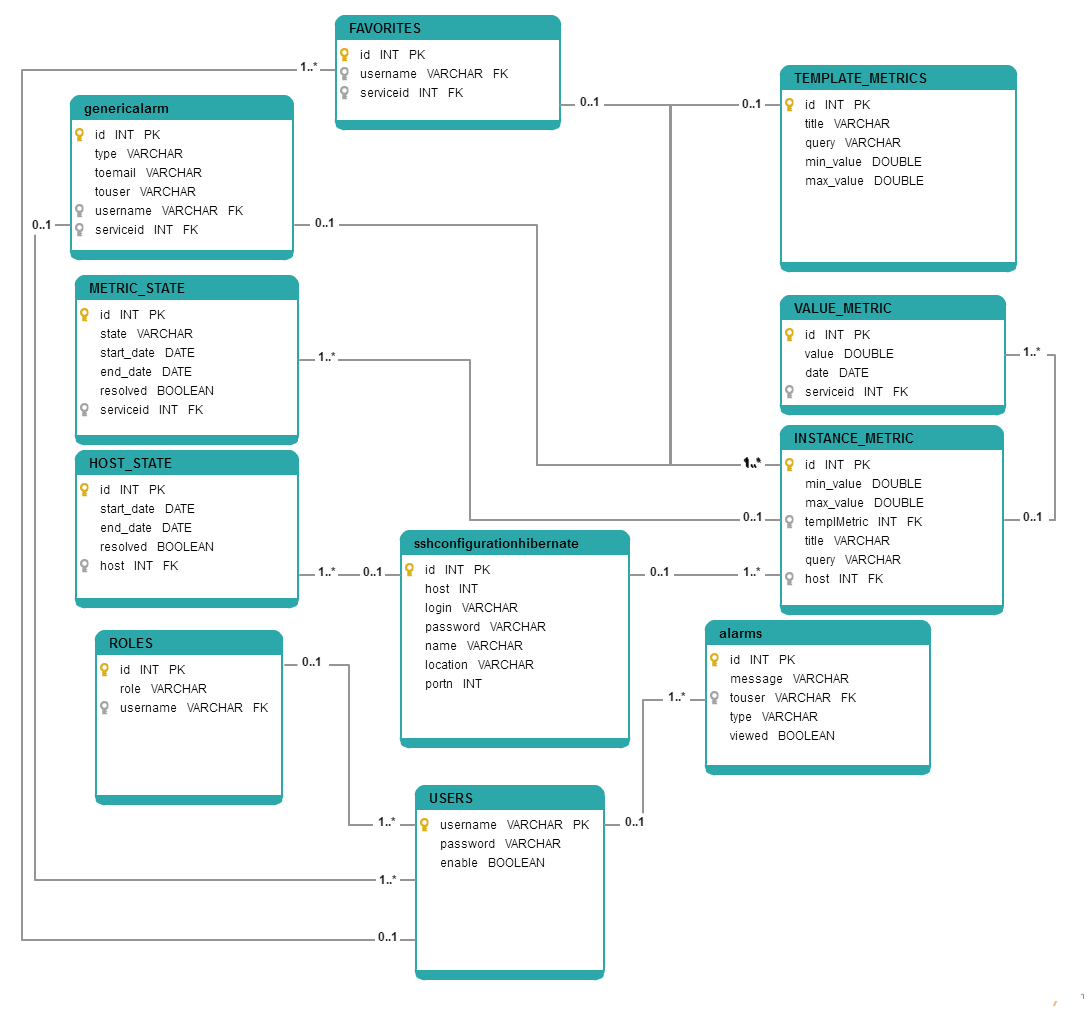


Рис.9 “Схема базы данных”

В системе предусмотрена защита ввода в адресную строку адреса недоступного для данного пользователи, а также “инжекции” в файл javascript функции, которая может получить данные доступные только администратору. При попытки доступа к данным обычным пользователем, он будет перенаправлен на страницу с кодом ошибки 404 (Рис.10).

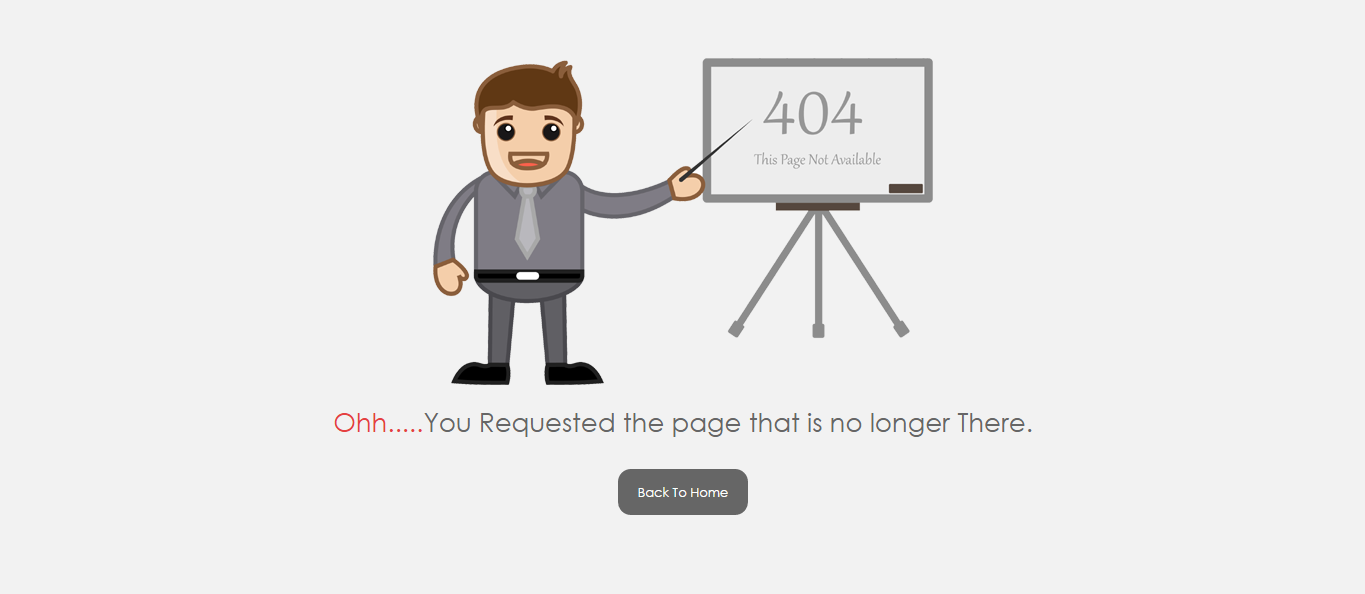


Рис.10 “Страница 404”

Данная ошибка не отражает ошибку связанную с правами доступа, но такой подход был выбран для того, что бы пользователь не мог точно знать существует ли данная функция на сервере.

Данная возможность реализована с помощью файла конфигурации, который был унаследован от WebSecurityConfigurerAdapter [2].

Также на UI пользователя скрыто меню администратора, сделано это с помощью зависимости thymeleaf-extras-springsecurity4 и тега <div sec:authorize="hasRole('ROLE\_ADMIN')"> </div>. Содержимое данного тега отрисовывается только пользователю с ролью “ROLE\_ADMIN” (Рис.11).

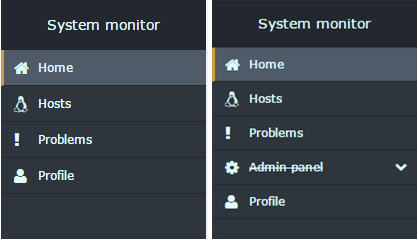


Рис.11 “Скрытое меню”

# Вывод:

Из всего вышесказанного можно сделать вывод, что данное приложение решает поставленные задачи. Данный сервис позволит вовремя получать необходимую информацию для того, чтобы принять нужное решение и устранить проблему.

**Список использованных источников**

1. Highcharts: Interactive JavaScript charts for your webpage. [Электронный ресурс]. – Режим доступа: http://www.highcharts.com, свободный. – Загл. с экрана.
2. Spring Security : Customize 403 access denied page. [Электронный ресурс]. – Режим доступа: http://www.mkyong.com/spring-security/customize-http-403-access-denied-page-in-spring-security, свободный. – Загл. с экрана.

**Аннотация**

Данная статья посвящена решению проблемы возможного выхода из строя сервера. В статье озвучены возможные причины появления проблемы и представлен способ их решения. В качестве решения представлен web сервис. Автор акцентирует внимание на детали реализации самого сервиса и на проблему связанную с хранением и предоставлением данных пользователю и объясняет выбранные им подходы. Представлена подробная схема базы данных и описание к каждой из таблиц. Большая часть статьи выделена под взаимодействие с пользователем, с предоставленными ему возможностями с разделением на роли. Также в статье присутствует описание решение проблемы умышленной попытки доступа обычного пользователя к данным открытым только для роли администратора. В качестве примера в статье используются несколько виртуальных машин с предустановленными на них ubuntu 14.04 и представлены скриншоты с данными, которые были считаны с данных виртуальных машин.