Удалённый мониторинг системы

# Задачи:

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

# Вступление:

Многие предприятия приобретают большое количество ПК и нанимают к себе на работу системного администратора (СА), в обязанности которого входим контроль работы всех ПК предприятия. На предприятии многие ПК работают по 24 часа в сутки, СА не может следить за данными ПК всё время, сбой в работе может произойти и в ночное время, что очень негативно повлияет на работу системы, а может и всего предприятия. Многие проблемы можно решить ещё до их появления, в данном случаи необходимо ПО которое будет производить мониторинг системы круглосуточно и сообщать о превышении значения какой-либо метрики снятой с определённого ПК.

# Используемые технологии:

ПО реализовано на языке программирования Java с использованием Spring MVC, Hibernate Framework и highcharts. Для работы ПО необходима БД (строгого ограничения в СУБД нету), которая будет хранить всю информацию. Присутствует возможность использования двух разных БД, одна из которых будет хранить всю необходимую информацию о подключения к удалённым ПК, другая будет хранить большое количество данных снятых в процессе работы.

# 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:

• Добавление нового хоста

• Просмотр сведений о хосте

• Просмотр возникших проблем

• Система оповещения

• Графическое представление для состояния метрик

• Настройка графиков для главной страницы

Администратору предоставлены все возможности пользователя, а также следующие функции:

• Редактирование хостов

• Редактирование локальных метрик

• Редактирование шаблонных метрик

• Редактирование сервисов для хоста

• Добавление новой шаблонной метрики

• 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 управляющие жизненным циклом ПО.
4. DB – Слой базы данных, для хранения необходимой информации.

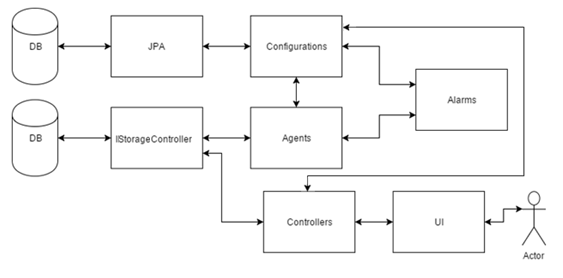


Рис.1 “Архитектура приложения”

Слой BD состоит из двух баз данных, с одной из которых взаимодействует реализация интерфейса IStorageController, она хранит в себе информацию о метриках и их значениях. Взаимодействие со второй BD происходит через интерфейс JPA, в этой базе хранятся информация о хостах и уведомлениях.

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

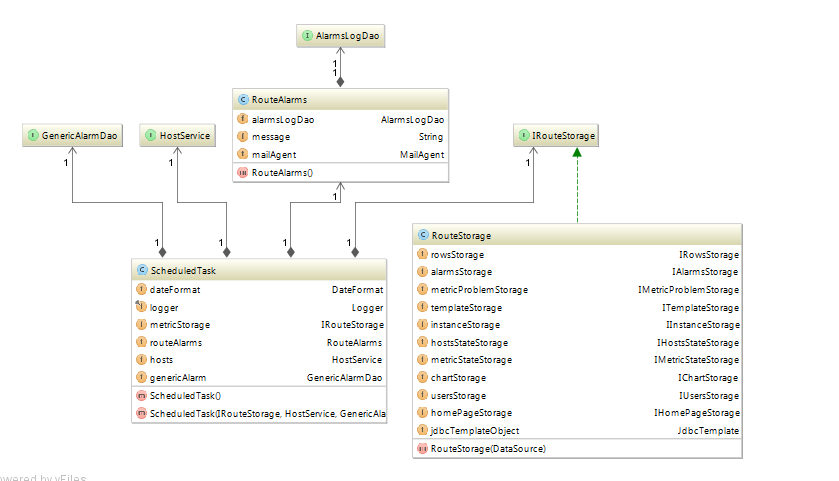


Рис.2 Диаграмма классов “ScheduledTask”

Метод run() поочерёдно считывает информацию о каждом хосте, которая хранится в базе, создаёт объект класса SSHAgent, который по сетевому протоколу SSH подключается к удалённому хосту (Рис.3).

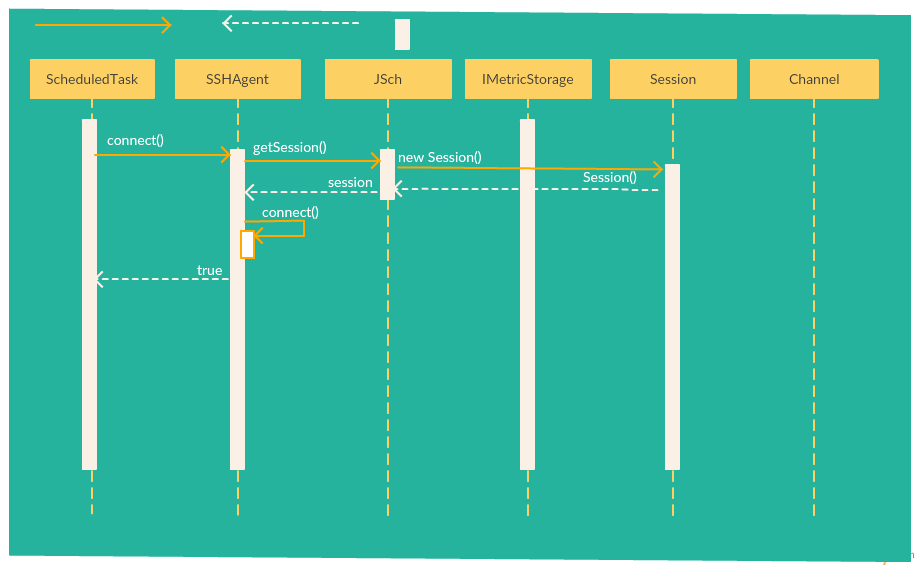


Рис.3 “Открытие сессии”

Если подключение не удалось установить, то с помощью сервиса “RouteAlarms” происходит отправка сообщения об ошибке (Рис.5),

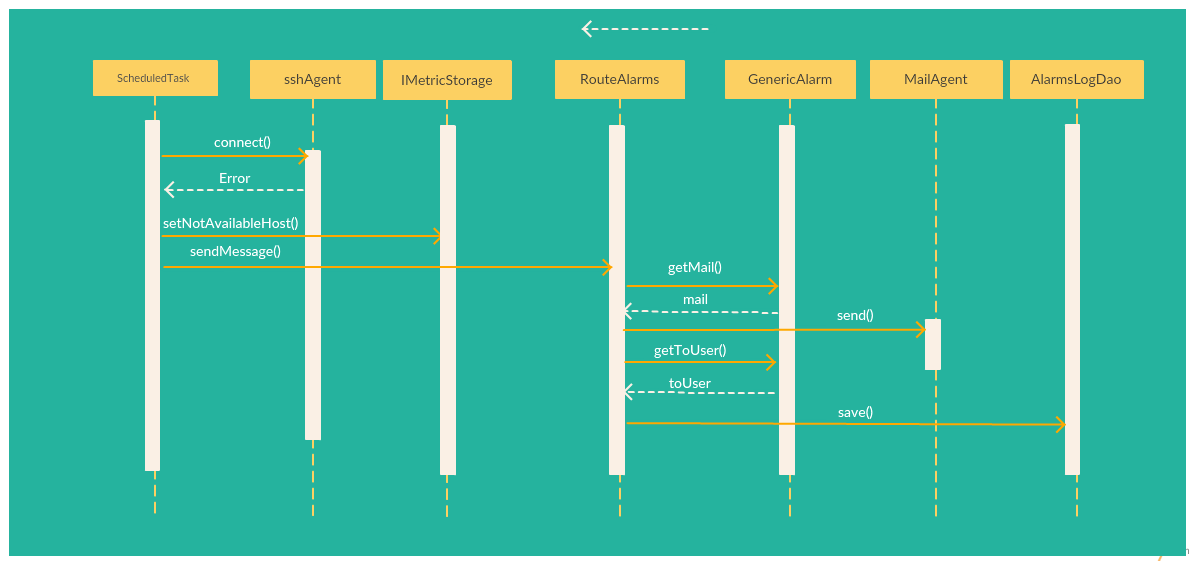


Рис.5 “Оповещение пользователя о проблеме с хостом”

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).

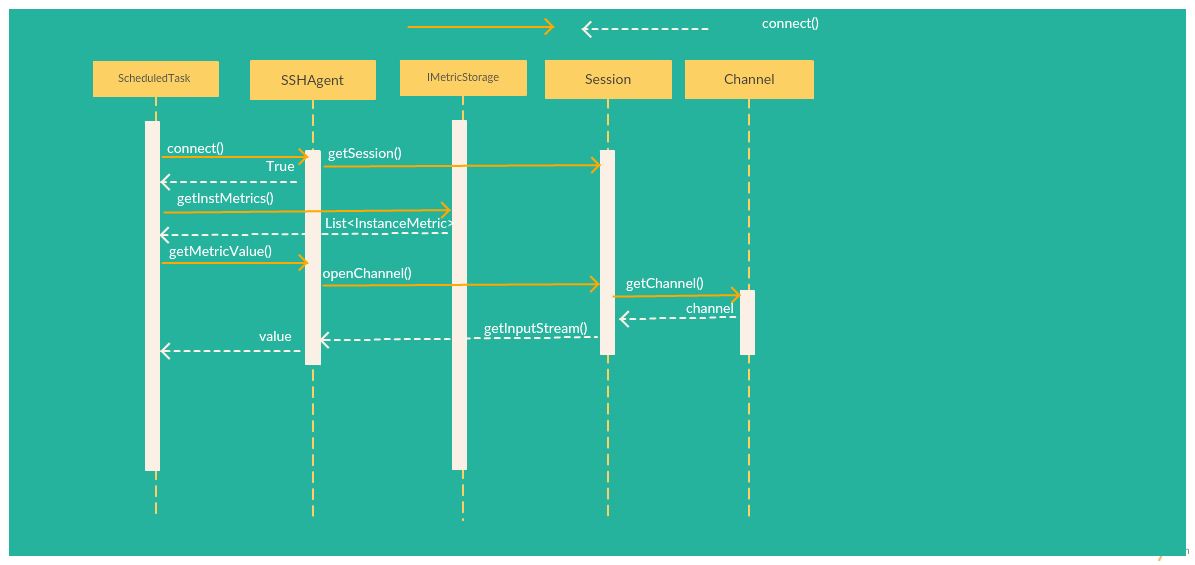


Рис.4 “Снятие значений метрик с хоста”

С помощью метода rageValue сервиса “ScheduledTask” сравнивается полученное значения с пороговыми записанными в InstanceMetric. Если значение выходит за определённое пороговое значение, записанное в InstanceMetric, то с помощью сервиса “RouteAlarms” происходит отправка сообщения об ошибке (Рис.7).

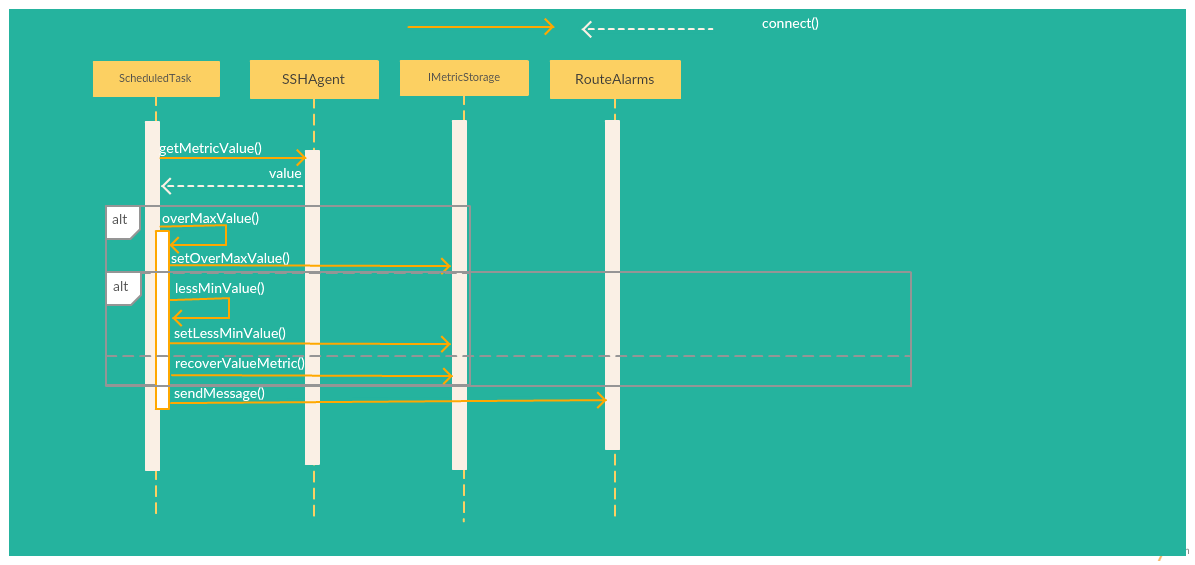


Рис.7 “Оповещение пользователя о состоянии метрики”

Далее цикл для сервиса “ScheduledTask” повторяется.

Для работы с БД Spring создаёт, хранит и инжектит в необходимые нам классы сервис “IStorageController”, который отвечает за логику работы с БД. Все действия пользователя через Web интерфейс обрабатываются определённым контроллером, который в свою очередь использует сервис “ IStorageController ”. Например, пользователь заходит на страницу со всеми проблемами, контроллер данной страницы идёт к Spring, просит у него предоставить объект StorageController который реализует интерфейс IStorageController и просит предоставить ему список проблем метрик List<MetricState> и хостов List<HostsState>, в свою очередь StorageController с помощью метода queryForList(String) объекта jdbcTemplateObject (который был создан при инициализации StorageController) посылает запрос в БД и на выходе принимает список List<Map<String, Object>>, который в дальнейшем обрабатывается и заполняет List<MetricState>, по завершению заполнения листа StorageController возвращает список проблем с метриками (для хостов аналогично) в котроллер, который добавляет объект в представление ModelAndView для дальнейшей обработки с помощью технологии thymeleaf, которая вернёт уже готовую страницу пользователю.

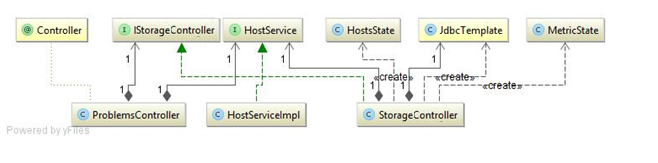


Рис.8(ссылку на рис. в тексте и название )

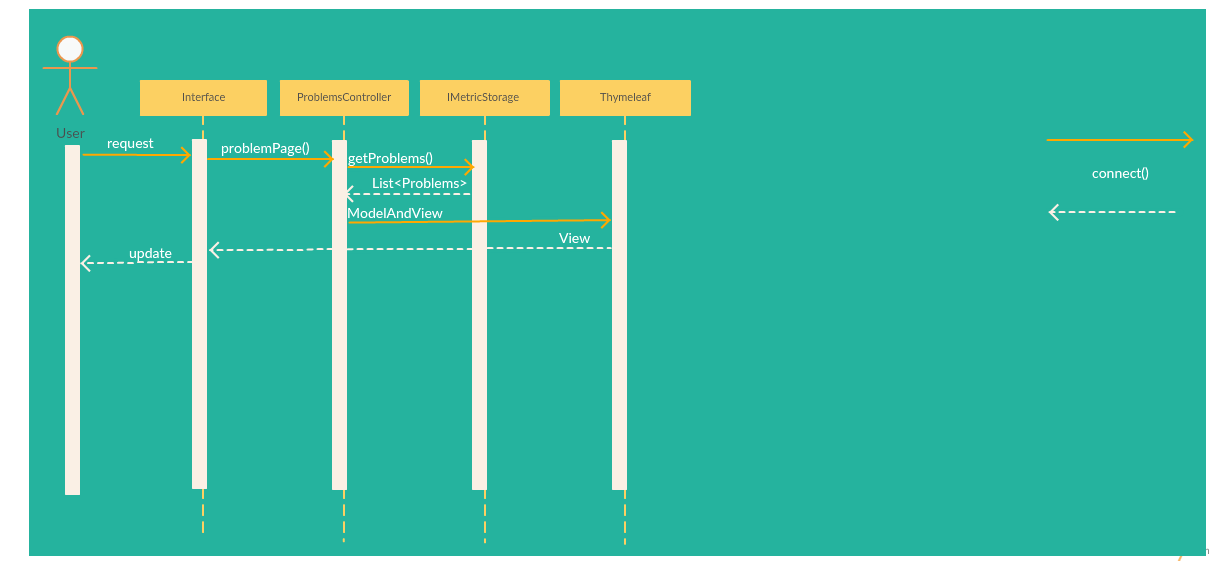


Рис.9 “Страница с проблемами” ссылку на рис. в тексте и название )

В приложении реализовано графическое представление для снятых значений метрик. Данная функция реализована с помощью Highcharts. Highcharts — библиотека для создания чартов написанная на языке JavaScript. Для получения необходимых данных для построения графика используется ajax запрос, который передаёт id метрики и дату (если дата не указана, то построение графика будет производиться от последнего снятого значения) в контроллер. После контроллер обращается в IChartStorage, который в свою очередь посылает запрос в базу данных и принимает последовательность значений метрики за определённый период времени. Так, как количество ‘точек’ за некоторый период может быть очень большим, то необходим агрегация данных. На выбор предоставляется несколько “масштабов”, а именно 1 минута, 3 минуты, час, день, неделя, месяц, 6 месяцев, год и все значения, агрегация производится для “масштаба” час, день, неделя, месяц, 6 месяцев. Для масштаба 1 и 3 минуты сравнивается разница времени между соседними точками, пока разница больше чем 10 секунд в объект Map<Long, Object> записывается null. Для агрегации масштаба высчитывается среднее арифметическое за предыдущий “уровень” масштаба и производится запись этого значения и времени в объект Map<Long, Object>, если метрика на данной ед. масштаба не имеет значений в базе, то вместо значения записывается null, это необходимо для обрыва графика в данном месте. После агрегации объект Map<Long, Object> передаётся на обработку в javascript где поточечно производится отрисовка графика. При необходимости пользователь может поменять масштаб графика с помощью наведения на нужный график курсора мыши и прокруткой колёсика, при прокрутке 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.

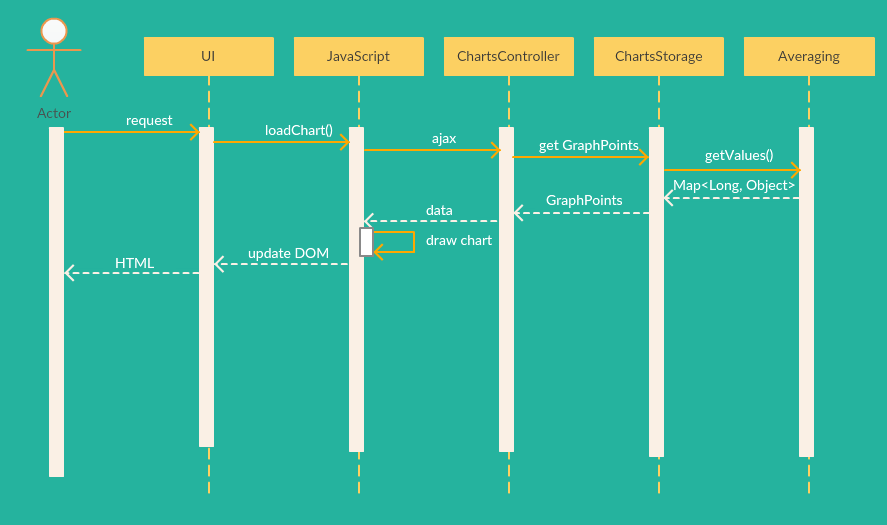


Рис.10 “Отрисовка графика” ссылку на рис. в тексте и название )

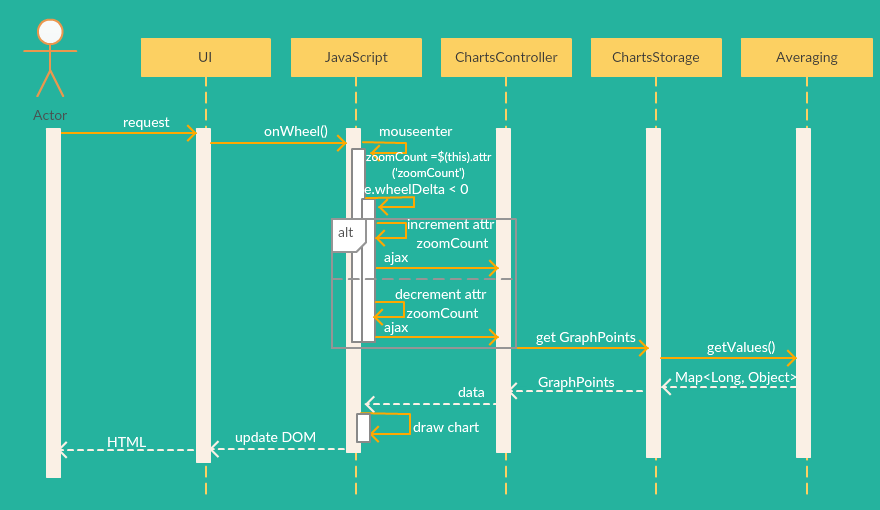


Рис.11 “Масштабирование графика” ссылку на рис. в тексте и название )

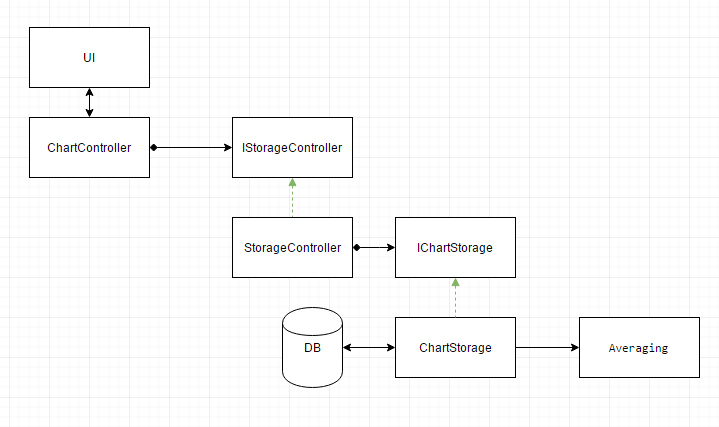


Рис.12 “Диаграмма классов для отрисовки графика”

Логика работы бд подразумевает использование двух различных баз данных.

База данных с которой взаимодействует реализация интерфейса IStorageController содержит такие таблицы, как:

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 со второй БД происходит с помощью библиотеки hibernate, данная база данных хранит такие таблицы, как:

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.

Схему этих двух баз данных можно представить в виде одной базы данных. На данной схеме все лини связи подразумевают тип связи “один ко многим”.

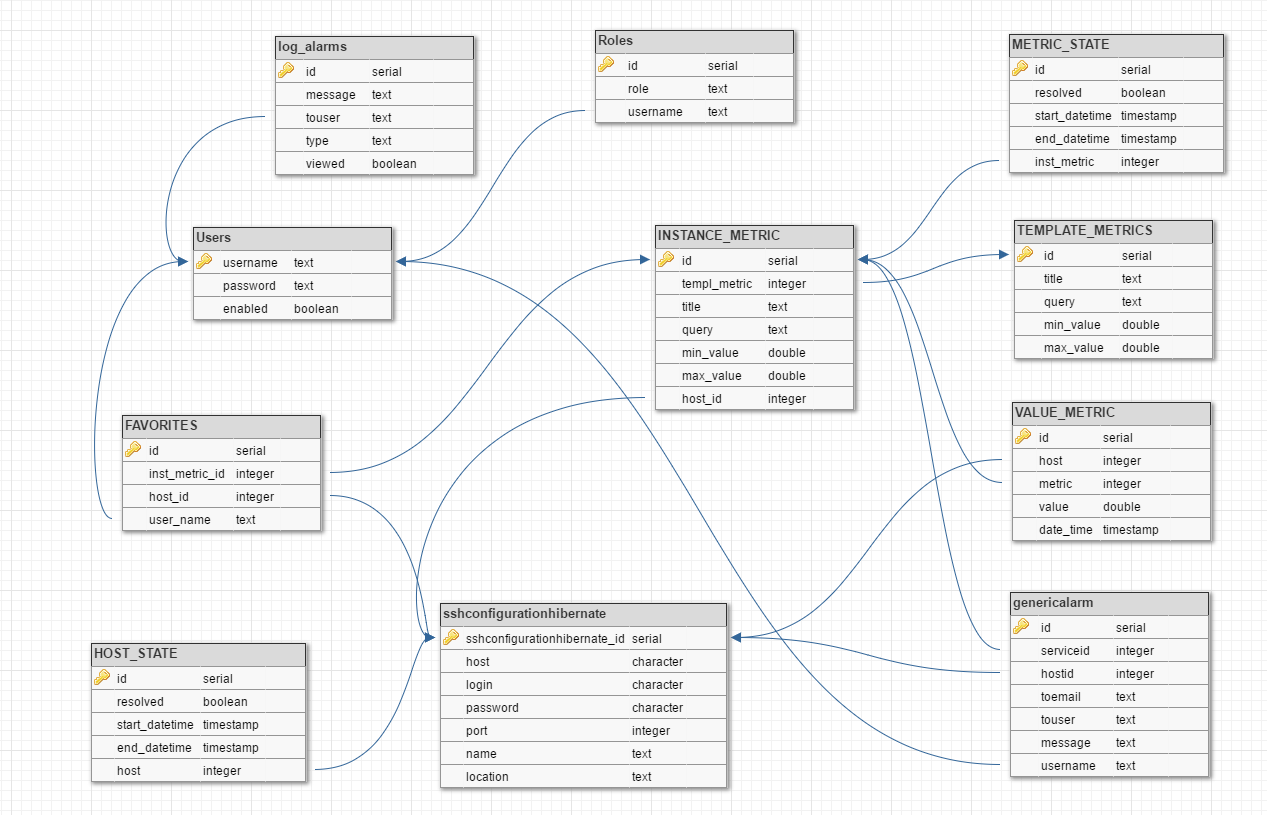


Рис.13 “Схема базы данных”

В системе предусмотрена защита ввода в адресную строку адреса недоступного для данного пользователи или “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, хотя 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 и содержит следующую строку: “http.authorizeRequests().antMatchers("/admin/\*\*").access("hasRole('ROLE\_ADMIN')")…and().exceptionHandling().accessDeniedPage("/403");”. Данная строка конфигурация позволяет все пользователей, которые не являются администраторами перенаправлять по адресу “/403”. 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”.

# 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.