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 запускается метод run() сервиса “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.2), если подключение не удалось установить, то с помощью сервиса “RouteAlarms” происходит отправка сообщения об ошибке (Рис.4), если подключение удалось установить, то IMetricStorage возвращает все InstanceMetric для данного хоста и для каждой метрики SSHAgent создаёт новый канал, отправляет команду считавшую из поля в InstanceMetric и возвращает double значения (значения метрики в данный момент) (Рис.3). С помощью метода rageValue сервиса “ScheduledTask” сравнивается полученное значения с пороговыми записанными в InstanceMetric и если значение выходит за порог, то с помощью сервиса “RouteAlarms” происходит отправка сообщения об ошибке (Рис.6). Далее цикл для сервиса “ScheduledTask” повторяется. Диаграмма классов для сервиса “ScheduledTask” представлена на Рис.1

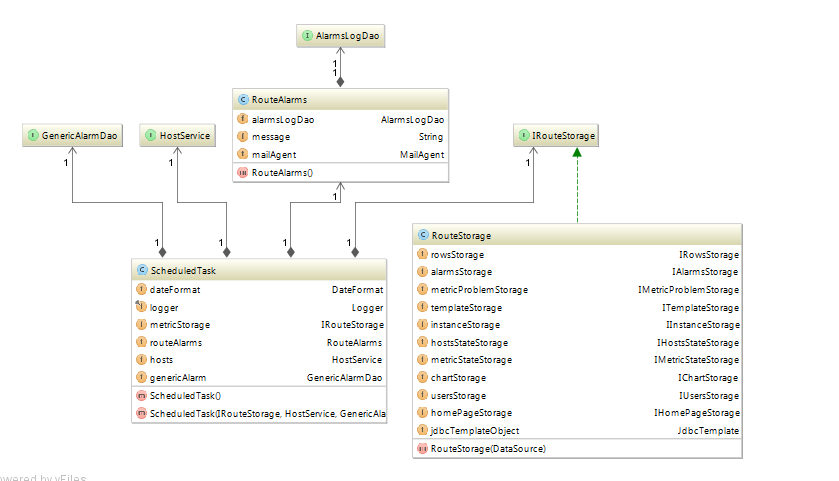


Рис.1 “ScheduledTask”

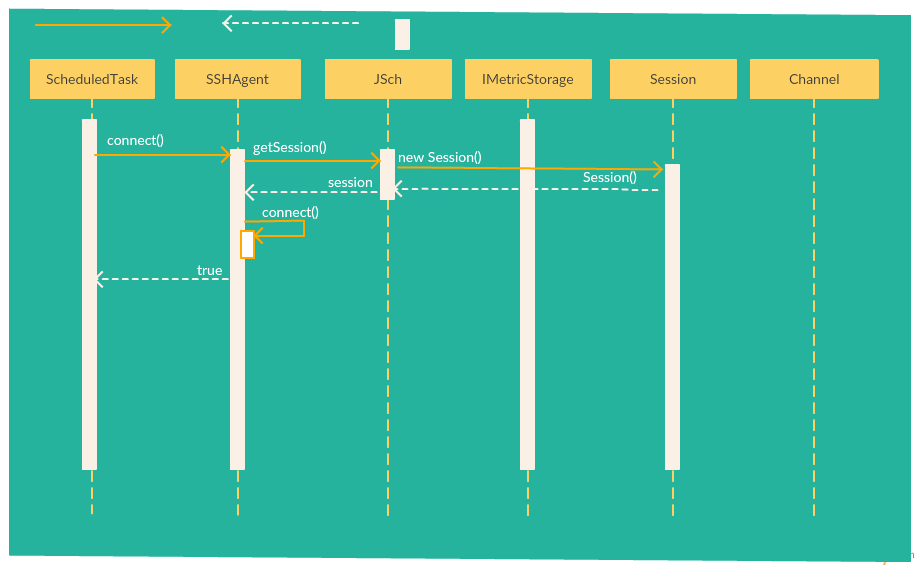


Рис.2 “Открытие сессии”

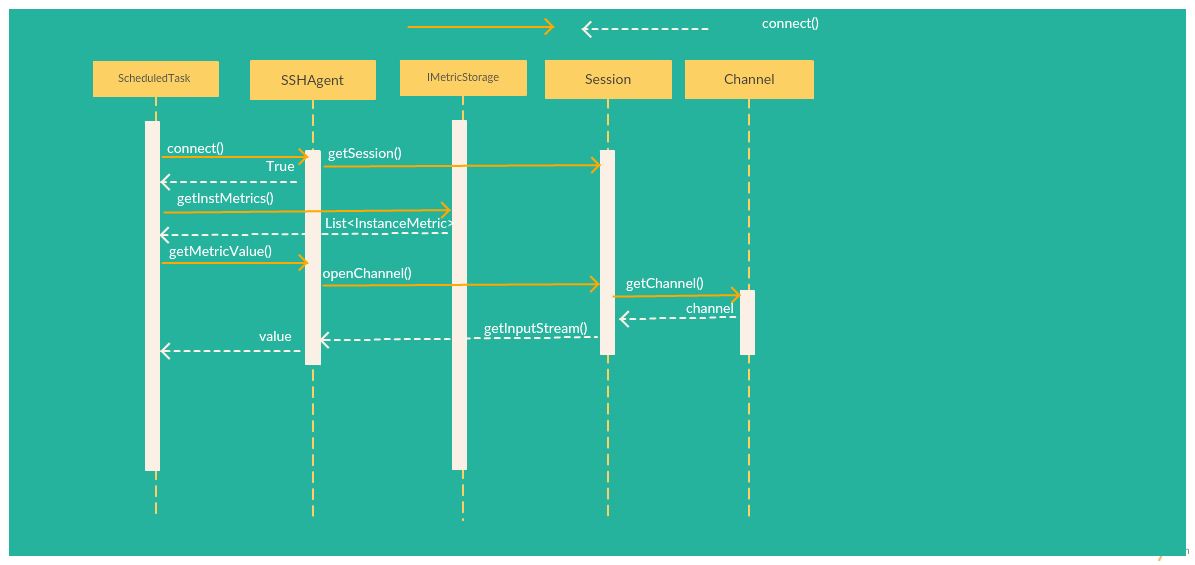


Рис.3 “Снятие значений метрик с хоста”

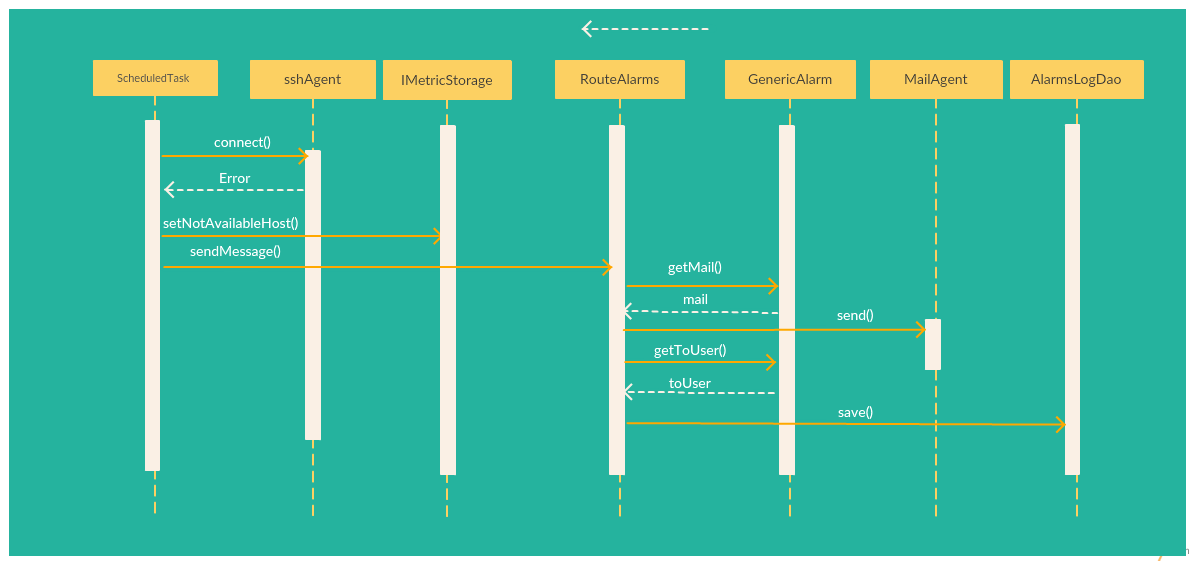


Рис.4 “Оповещение пользователя о проблеме с хостом”

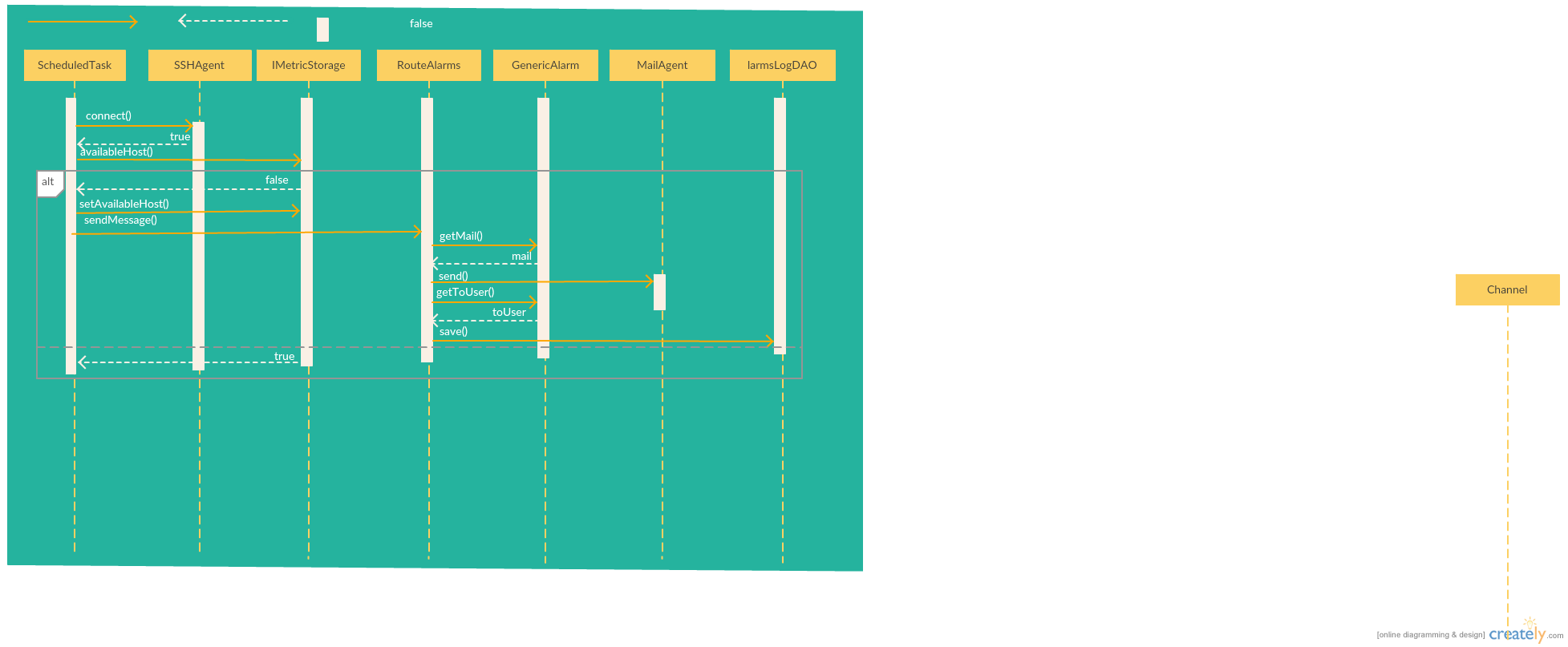


Рис.5 “Оповещение пользователя о работоспособности хоста”

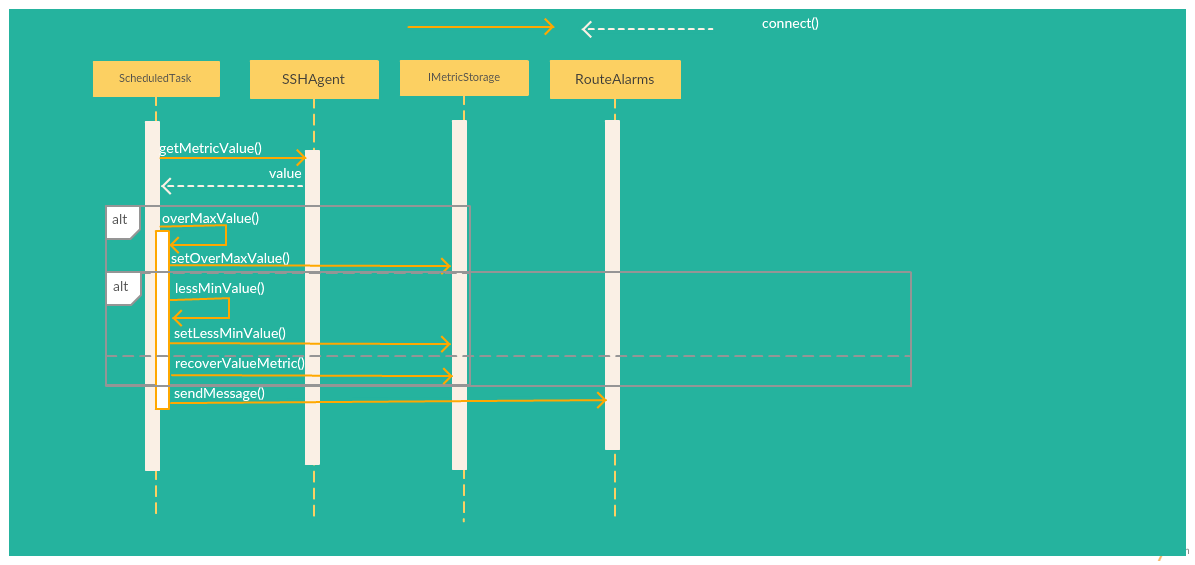


Рис.6 “Оповещение пользователя о состоянии метрики”

Для работы с БД Spring создаёт, хранит и инжектит в необходимые нам классы сервис “IMetricStorage”, который отвечает за логику работы с БД. Все действия пользователя через Web интерфейс обрабатываются определённым контроллером, который в свою очередь использует сервис “IMetricStorage”. Например пользователь заходит на страницу со всеми проблемами, контроллер данной страницы идёт к Spring, просит у него предоставить объект MetricStorage который реализует интерфейс IMetricStorage и просит предоставить ему список проблем метрик List<MetricState> и хостов List<HostsState>, в свою очередь MetricStorage с помощью метода queryForList(String) объекта jdbcTemplateObject (который был создан при инициализации MetricStorage) 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 MetricStorage 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.

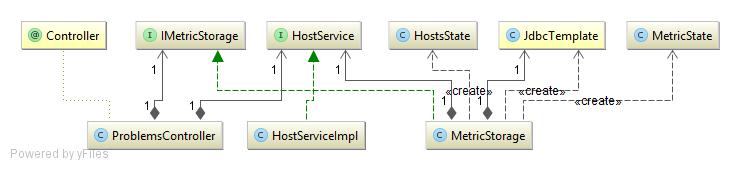


Рис.7

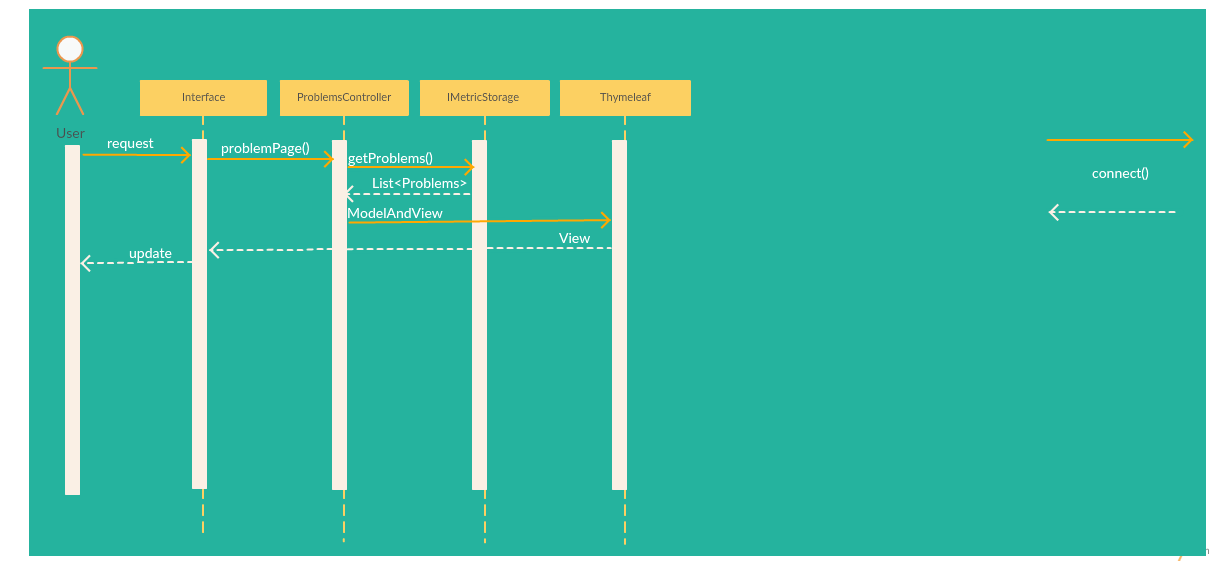


Рис.8 “Страница с проблемами”

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.

Для агрегации масштаба “час” высчитывается среднее арифметическое за 1 минуту и производится запись этого значения и времени в объект Map<Long, Object>, если метрика в данную минуту не снималась, то в поле значение записывается null, это необходимо для обрыва графика в данном месте. Для агрегации масштаба “дня” соответственно добавиться ещё и среднее арифметическое часа.