



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА


К КУРСОВОМУ ПРОЕКТУ

по дисциплине: «Технологии Интернет»

НА ТЕМУ:

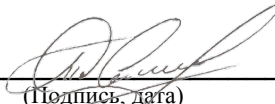
**«Разработка веб-ориентированной подсистемы
ведения журнала сообщений о событиях в системе»**

Студент РК6-83
(Группа)


(Подпись, дата)


В.О. Голубев
(И.О.Фамилия)

Руководитель курсовой работы


(Подпись, дата)

А.П. Соколов
(И.О.Фамилия)

Консультант


(Подпись, дата)

А. Ю. Першин
(И.О.Фамилия)

ПРОСМОТРЕНО

Соколов А.П. 18:19, 17/5/20

2020 г.

**Рекомендованная
оценка отлично!**

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____

(Индекс)

_____ А.П. Карпенко _____

(И.О.Фамилия)

« _____ » _____ 20__ г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Технологии Интернет

Студент группы РК6-83

_____ Голубев Владислав Олегович _____

(Фамилия, имя, отчество)

Тема курсовой работы Разработка веб-ориентированной подсистемы ведения журнала сообщений о событиях в системе

Направленность КР (учебная, исследовательская, практическая, производственная, др.)

_____ практическая _____

Источник тематики (кафедра, предприятие, НИР) _____ Кафедра _____

График выполнения КР: 25% к 4 нед., 50% к 8 нед., 75% к 12 нед., 100% к 17 нед.

Техническое задание Создать программную инфраструктуру для ведения журнала о событиях в системе и разработать пользовательский интерфейс для показа сообщений

Оформление курсовой работы:

Расчетно-пояснительная записка на 21 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

_____ 7 рисунков _____

Дата выдачи задания « 18 » февраля 2020 г.

Руководитель курсовой работы

_____ (Подпись, дата)

_____ А.П. Соколов _____

(И.О.Фамилия)

Студент

_____ (Подпись, дата)

_____ В.О. Голубев _____

(И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту,
второй хранится на кафедре.

АННОТАЦИЯ

Пояснительная записка оформлена в рамках выполнения курсового проекта по курсу «Технологии Интернет». В записке дан обзор на существующие подходы к логированию событий в вычислительных системах и уведомлению пользователей о возникающих событиях. Описано создание программной инфраструктуры подсистемы журнала сообщений для PBC GCD, описание программной архитектуры сервера приложений и протокола сообщения между клиентом и сервером приложений в PBC GCD.

Тип работы: курсовая работа

Тема работы: Разработка веб-ориентированной подсистемы ведения журнала сообщений о событиях в системе.

Объект исследований: системы логирования событий в веб-приложениях.

СОКРАЩЕНИЯ

PBC – Распределенная вычислительная система.

САПР – Система автоматизированного проектирования.

DOM – Document Object Layer.

MVC — Model-View-Controller.

CRUD — Create-Read-Update-Delete.

IPC — Inter-Process Communication.

СОДЕРЖАНИЕ

АННОТАЦИЯ	3
СОКРАЩЕНИЯ.....	4
СОДЕРЖАНИЕ	4
ВВЕДЕНИЕ	5
1. ПОСТАНОВКА ЗАДАЧИ	8
1.1. Концептуальная постановка задачи	8
2. АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ.....	9
2.1. Общая схема узлов PBC GCD.....	9
2.2. Информационная модель подсистемы ведения журнала сообщений о событиях в системе.....	12
2.3. Программная архитектура подсистемы журнала сообщения в comwps	13
2.4. Программная архитектура подсистемы журнала сообщения в русomaps.....	15
2.4.1. Асинхронная обработка запросов	15
2.4.2. Протокол коммуникации с веб-клиентом	16
3. ТЕСТИРОВАНИЕ И ОТЛАДКА.....	18
4. АНАЛИЗ РЕЗУЛЬТАТОВ.....	20
ЗАКЛЮЧЕНИЕ	20
СПИСОК ЛИТЕРАТУРЫ	21

ВВЕДЕНИЕ

В современных сложных программных системах пользователь часто совершает ошибки в ходе работы. Даже если пользователь прошел обучение по работе с программой, это не гарантирует, что при попытке освоить более сложные сценарии для решения своих задач, он не допустит ошибок и программа отработает корректно. Нет, также, никаких гарантий, что при разработке сложной системы не было допущено ошибок в программном коде. При возникновении внештатных ситуаций в работе с программой пользователь нуждается в объяснении, почему он не смог получить нужный результат. Разработчикам получение этой информации необходимо для выявления ошибок в программе или документирования наиболее частых ошибок пользователя при работе с программой и способов их решения. В обоих случаях решением подобной проблемы является средство обратной связи от программы, которое может информировать о возникновении исключительных ситуаций и их причине. Таким средством является журнал сообщений о событиях в системе, также известный как система логирования.

Процесс логирования (журналирования, протоколирования) - автоматическая запись событий, происходящих в рамках определенного процесса. Логирование позволяет оценивать состояние процесса и регистрировать ошибки. Главное преимущество использования логирования заключается в предоставлении возможности контролировать процесс выполнения бизнес-логики как отдельного процесса, так и системы в целом. Грамотно реализованный процесс учета событий позволяет получать параметризованные данные логики работы приложения. Подобный инструмент может помочь разработчикам улучшать стабильность системы.

Сложность проектирования и разработки системы логирования зависит от сложности самой системы. Основными требованиями являются [1]:

- наличие централизованного хранилища данных,
- отображение текущего состояния системы и её отдельных процессов,

- регистрация событий на любых компьютерах и серверах вычислительной системы,
- наличие функциональной системы поиска и фильтрации зарегистрированных в системе событий,
- возможность централизованного администрирования и мониторинга системы,
- наличие системы оповещения о возникновении определенных событий.

Единое хранилище данных нужно для синхронизации с состоянием программы или целой вычислительной сети. Само хранилище может представлять собой как файл или систему файлов, так и целую базу данных в зависимости от требований к системе. Каждое событие должно быть зарегистрировано в хранилище в определенном формате. Формат должен быть удобен для интерпретации как на программном уровне, так и на уровне представления человеком. Для повышения удобства специалисты пришли к идее разделения типов сообщений на несколько уровней [2], [3]. Такое разделение помогает фильтровать логи приложений и настраивать систему логирования на запись сообщений только о необходимых событиях. Уровни представляют собой набор из одного или нескольких типов сообщений. К основным типам сообщений о событиях относят:

- ошибки,
- предупреждения,
- информация.

В зависимости от сложности системы каждый тип может быть разделен на несколько подтипов. Программный продукт на различных стадиях своего жизненного цикла нуждается в разных уровнях логирования. При создании и тестировании программы разработчикам важно иметь как можно больше информации о поведении системы. Для этого система логирования работает на всех трех уровнях и записывает сообщения всех типов событий. Во время эксплуатации системы пользователями большое количество операций записи

будет плохо сказываться на производительности системы, поэтому система логирования должна записывать только сообщения о критических ошибках.

В современных САПР важным элементом является своевременное оповещение пользователя о событиях, происходящих во время работы. Информация об успешном выполнении расчета, предупреждение о возможной некорректной работе системы с полученными данными или оповещение о возникшей ошибке позволяют пользователю реагировать на ситуацию и предпринимать соответствующие действия. Правильно спроектированная система оповещений дополняет пользовательский интерфейс для работы с программой и становится помощником для решения прикладных задач. Для достижения этой цели система оповещений должна быть тесно связана с системой логирования.

Современные решения систем оповещений основываются на технологии push-уведомлений [4], [5]. Доставка сообщений в системах push-уведомлений осуществляется по сети интернет, используя клиент-серверную модель. Клиент подписывается на рассылку push-уведомлений, а сервер публикации (publisher), по наступлению определенного события, осуществляет рассылку подписавшимся клиентам. Данная технология уже является стандартом в мобильных технологиях. Реализации клиентов также имеются в браузерах [6] и даже на уровне окружений рабочего стола в операционных системах для персональных компьютеров.

Разработка веб-ориентированной подсистемы ведения журнала сообщений предполагает создание единообразного подхода для разработчиков разных компонентов РВС к информированию пользователя или системы о возникающих событиях. Для реализации подсистемы были изучены труды по разработке систем логирования и уведомлений. В статье Винничек Е. [1] рассмотрены основные проблемы и способы их решения при разработке систем логирования в сложных распределенных системах. Автор описывает проектирование системы логирования, которую можно было бы встроить в существующие РВС и показывает основные преимущества выбранного

подхода. Были изучены документации к операционным системам Microsoft [2] и man для unix-подобных операционных систем [3] на предмет реализации систем логирования. Была найдена информация о поддерживаемых типах сообщений о событиях в системе. В работе Федченко И. О. и Намиота Д. Е. [4] дано описание технологии push-уведомлений, рассмотрены существующие решения серверов публикации, а также приведен пример реализации информационной системы для мобильных пользователей, которая позволила бы создавать системы рассылок уведомлений без программирования.

Целью работы является разработка программной реализации подсистемы журнала сообщений о событиях в системе для PBC GCD. Необходимо реализовать поддержку подсистемы на сервере приложений русомарс и веб-клиенте comwps.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Концептуальная постановка задачи

Объект разработки: Подсистема ведения журнала сообщений о событиях в системе.

Цель разработки: Создание программного обеспечения для создания, интерпретации и визуализации сообщений о событиях в системе.

Задачи курсовой работы:

- 1) разработать программную реализацию для регистрации событий в системе и подготовки сообщения для выдачи пользователю,
- 2) разработать графический пользовательский интерфейс для просмотра и управления уведомлениями,
- 3) разработать протокол коммуникации между сервером приложений русомарс и веб-клиентом comwps в части сообщений об ошибках на стороне сервера.

Требования:

- 1) каждое сообщение должно быть следствием возникновения события в системе, которое должно быть зарегистрировано;

- 2) каждое сообщение, выводимое пользователю, должно быть связано с текущим сеансом Django работы данного пользователя;
- 3) подсистема вывода сообщений должна иметь возможность вывести широковещательное сообщение всем подключенным в данный момент пользователям;
- 4) для вывода сообщения должен быть создан удобный программный интерфейс, представленный в виде функции, которая на вход получает идентификатор требуемого для вывода сообщения, язык вывода, а также массив переменной длины значений параметров данного сообщения;
- 5) вывод сообщений пользователю должен осуществляться в специальное окно в web-клиенте;
- 6) сообщения разных типов должны сопровождаться соответствующей иллюстрацией.

2. АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ

Для реализации подсистемы необходимо разрабатывать «функционал» на сервере приложений и на веб-клиенте. Для написания серверной логики используется язык Python 3-й версии в связи с библиотекой для программирования сетевых приложений pyZMQ – реализации технологии ZMQ на языке Python. Для программирования на веб-клиенте используется язык Python в связи с веб-фреймворком Django, JavaScript в связи с библиотекой JQuery для манипулирования DOM и создания стандартных анимаций в пользовательском интерфейсе.

2.1. Общая схема узлов PBC GCD

На рисунке 1 представлена общая схема взаимодействия компонентов в PBC GCD, которая работает на базе клиент-серверной архитектуры. Веб-клиент comrws построен на веб-фреймворке Django. Django хорошо подходит для разработки веб-приложений разной сложности. Внутри он реализует паттерн MVC. Веб-клиент comrws использует собственный подход к разработке и в отличие от предлагаемого Django по-умолчанию синхронного режима обмена

информации по HTTP, использует технологию асинхронных запросов AJAX. Таким образом, comwps является нетипичным для веб-приложений на Django SPA-приложением, т.е. приложением, которое не требует перезагрузки в браузере при переходе на новую страницу. Веб-клиент общается с единой для РВС базой данных используя сервер somaps, который не представлен на рисунке 1, в качестве проксирующего сервера. Сам процесс коммуникации происходит по специальному протоколу. Передача информации по сети происходит по протоколу TCP. На рисунке 2 в виде UML-диаграммы представлена упрощенная последовательность обработки типичного запроса к comwps.

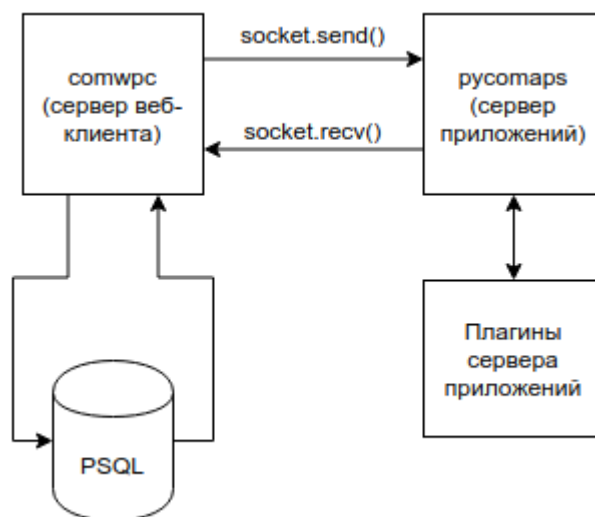


Рисунок 1 - Общая схема компонентов РВС GCD, в которых реализуется подсистема журнала сообщений о событиях

Связь с сервером приложений русомaps происходит, также, по TCP с помощью ZMQ-сокетов. Более подробно об этой технологии будет рассказано в разделе 2.4.1. Сервер приложений русомaps является новой версией сервера somaps. Общение с ним поддерживает всего несколько функций веб-клиента. Он написан на языке Python и для работы с сетью использует библиотеку pyzmq.

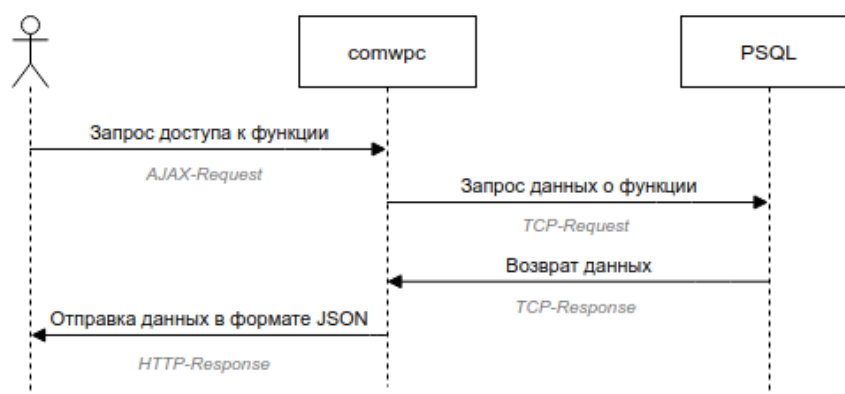


Рисунок 2 – UML-диаграмма последовательности обработки запроса от клиента

На рисунке 3 представлена общая схема взаимодействия сервера приложений с множеством клиентов. Обработчик запросов на сервере представляет из себя связку ROUTER-DEALER. ROUTER собирает запросы от клиентов. DEALER распространяет собранные запросы от клиентов на процессы обработки — Worker. Ответ от Worker идет через DEALER на ROUTER, а оттуда попадает нужному клиенту. Worker-ы являются плагинами к серверу приложений, которые осуществляют непосредственную логику работу функций PBC GCD. Например, плагин под названием GRPH_SOLVER_HANDLER_WEB осуществляет удаленный запуск произвольного графоориентированного решателя.

Сервер приложений русомарс реализует уникальный подход к асинхронной обработке запросов от клиентов. Подробнее об этом написано в разделе 2.4.1.



Рисунок 3 – Схема паттерна ROUTER-DEALER в русомарс

2.2. Информационная модель подсистемы ведения журнала сообщений о событиях в системе

На рисунке 4 представлена реляционная модель данных в базе данных, которая определяет логику работы с системными уведомлениями. Главной является таблица «Стандартные сообщения». В ней находятся уникальный идентификатор формата сообщения и строковое поле формата сообщения. Формат сообщения – это форматная строка, которая обычно используется для вывода форматных сообщений в языках C/C++. Чтобы сформировать итоговое сообщение для лога, в форматную строку подставляются параметры (численные или строковые). Каждое стандартное сообщение имеет свой тип, язык вывода и метод вывода (в каком узле РВС сообщение должно выводиться и каким образом). В РВС GCD поддерживаются два языка вывода: русский и английский.

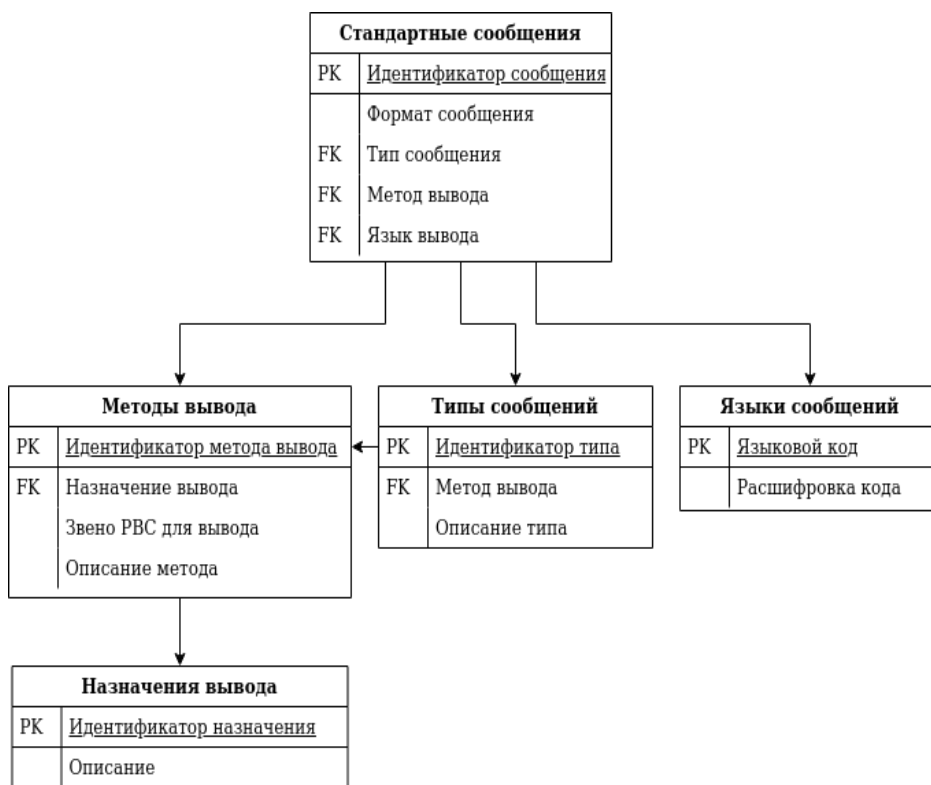


Рисунок 4 - Информационная модель подсистемы журнала системных событий

2.3. Программная архитектура подсистемы журнала сообщения в comwrc

В результате исследований по теме и требованиям к разработке было решено, что каждое событие в системе должно быть зарегистрировано в едином

хранилище. Это верно для критических ошибок в системе на стороне сервера приложений. Если возникает ошибка в результате обработки запроса от пользователя, то такие события важны только для пользователя и информация о них является временной, поэтому было решено, что пользовательские события должны храниться во временном хранилище на стороне клиента. Такое решение, также, обусловлено требованием связывать возникающее событие с сессией текущего пользователя.

Для регистрации события была разработана Django-модель Events. Django-модель представляет собой объектно-ориентированную обертку над инфологической единицей данных. Все CRUD-операции с объектом модели транслируются на базу данных. В качестве базы данных сервер веб-клиента использует SQLite. Методы класса модели Events реализуют функции регистрации и обработки события, а также подготовки данных для отображения сообщения пользователю. На рисунке 5 представлена блок-схема обработки возникшего на стороне сервера приложений пользовательского события в веб-клиенте.

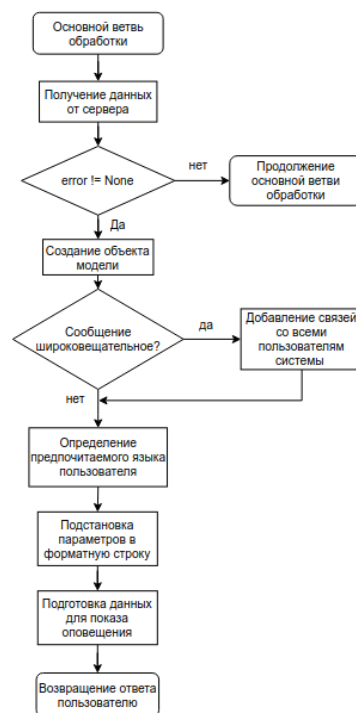


Рисунок 5 — Блок-схема алгоритма обработки возникшего события

Для разработчиков был разработан интерфейс в виде двух публичных методов класса Events. Метод `Events.registr(event_msg, user, is_broadcast)`

позволяет зарегистрировать событие `event_msg` и связать его с пользователем `user`. Опционально можно создать ширковещательное сообщение, если передать параметр `is_broadcast=True`. Метод `Events.get_message()` возвращает словарь с готовыми полями для отображения данных пользователю. Сообщение будет выдано на языке, который указан у пользователя предпочтительным в браузере. Это достигается за счет анализа полей HTTP-запроса.

Для показа сообщений были разработаны функции на языке JavaScript, которые показывают всплывающее сообщение, а также функции для управления оповещениями в панели уведомлений. Пример работы этих функций показан в разделе 3.

2.4. Программная архитектура подсистемы журнала сообщения в русомарс

2.4.1. Асинхронная обработка запросов

Технология ZMQ представляет API для удобной работы с сокетами, который реализует наиболее востребованные паттерны работы в сетевом программировании. Например, самый распространенный паттерн Request-Reply представлен специальной парой сокетов REQ и REP. ZMQ-сокеты являются усовершенствованной версией обычных unix-сокетов. Для работы с ними в библиотеке имеются соответствующие структуры данных и функции, которые берут на себя сложную работу с очередями сообщений, осуществляя обработку запросов без блокировок. Для этого в ZMQ очереди строятся на неблокируемых структурах данных [8]. Благодаря такому подходу работа с сокетами упрощается до вызова нескольких методов для передачи и принятия сообщений.

Логика обработки запросов в сервере приложений реализуется в классе `ApplicationServer`. Методы этого класса запускают сервер с нужной конфигурацией. На рисунке 3 представлена схема паттерна ROUTER-DEALER, который реализуется в этом классе для обработки запросов. На запрос от клиента создается новый процесс `Worker`, который выполняет определенную задачу. Каждый `Worker` может взаимодействовать с `ApplicationServer` с помощью IPC. Для этого создаются пары сокетов DEALER-REP. Самый

простой случай взаимодействия процессов `ApplicationServer` и `Worker` рассмотрен на рисунке 6. Можно увидеть, как достигается асинхронная обработка запросов от веб-клиента к `Worker`. Каждый процесс `Worker` состоит из двух параллельных потоков. `WorkerThread` является рабочим потоком, а `RequestHandler` — управляющим.

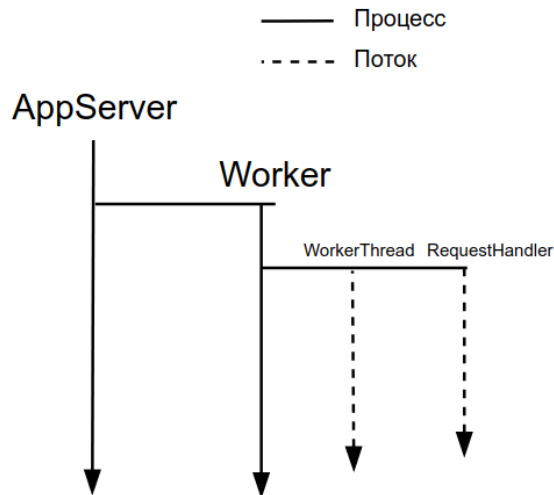


Рисунок 6 - Схема распараллеливания процессов `AppServer` и `Worker`

2.4.2. Протокол коммуникации с веб-клиентом

Текстовый протокол основывается на формате JSON. Пример сообщения на запуск обработчика на сервере приложений представлен в Листинге 1.

Листинг 1 Пример сообщения на запуск обработчика на стороне сервера приложений

```
{
  'command': 'create_worker',
  'worker_name': data['processing'],
  'input': {'data': data['data']}
}
```

Здесь в значении ключа `command` находится название команды, которую сервер приложений должен исполнить, `worker_name` — название обработчика запроса, `input` — поле, в котором хранится информация с передаваемыми данными.

Сериализация JSON в поток байт производится функцией стандартного модуля `json` языка Python. Десериализация на стороне сервера приложений производится, также, модулем `json`.

Кроме команды на создание обработчика, сервер приложений поддерживает команду запроса текущего статуса работы обработчика —

get_worker_result. При этом необходимо обязательно указывать идентификатор обработчика в теле запроса.

Обычно после запроса к самому серверу приложений далее производятся запросы к созданному обработчику, чтобы запросить у него какие-то данные или попросить выполнить дополнительную обработку с новыми данными. Для этого в контексте запроса обязательно должен быть указан идентификатор обработчика, а в теле запроса команда для обработчика и входные данные.

Ответ от сервера четко стандартизован. Это должен быть JSON с полями result и error. Поле result определяется для каждого Worker отдельно. Поле error стандартизовано для всех ответов сервера в случае ошибки, как показано на рисунке 7.

```
reply = {  
  'result': None,  
  'error': {  
    'id': 1,  
    'params': {  
      'ru': ['текст сообщения'],  
      'en': ['message text']  
    }  
  }  
}
```

Рисунок 7 – Формат ответа от сервера приложений в случае ошибки обработки запроса

Здесь поле id является номером стандартного сообщения из таблицы «Стандартные сообщения» информационной модели подсистемы ведения журнала системных событий. Поле params является словарем с ключами, которые являются языковыми кодами. По каждому ключу можно получить массив с параметрами для подстановки в форматную строку стандартного сообщения на соответствующем языке.

3. ТЕСТИРОВАНИЕ И ОТЛАДКА

Тестирование подсистемы ведения журнала сообщений проводилось на функции GRPH_SOLVER_WEB, которая производит запуск произвольного графоориентированного решателя. На рисунке 8 представлен графический пользовательский интерфейс на веб-клиенте для запуска этой функции.

Исходные данные для запуска функции заполняются в форме. Обязательным полем для запуска решателя является «Идентификатор решателя». Если указан неверный идентификатор, то на сервере должна произойти ошибка при запуске функции.

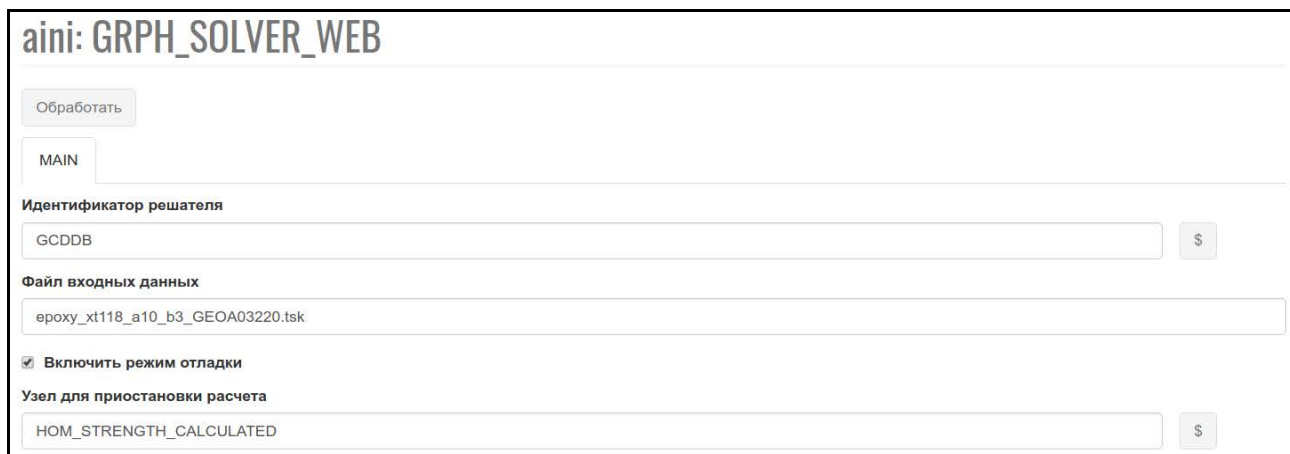


Рисунок 8 — Интерфейс для запуска функции GRPH_SOLVER_WEB

При нажатии на кнопку «Обработать» был отправлен запрос на сервер приложений для запуска плагина, который должен осуществить запуск решателя с заданными идентификатором. Поскольку идентификатор был указан неверный, плагин не смог найти файл с описанием графовой структуры решателя и вернул на клиент ошибку. Сообщение об ошибке было зарегистрировано на веб-клиенте и сформировано сообщение для пользователя на языке, который указан предпочтительным в браузере — русском. Сообщение показано на рисунке 9. Оно представляет из себя всплывающее окно. В заголовке окна обозначен тип возникшего события, а в тексте скомпилированная строка стандартного сообщения с текстовым параметром «Не удалось найти заданный решатель».

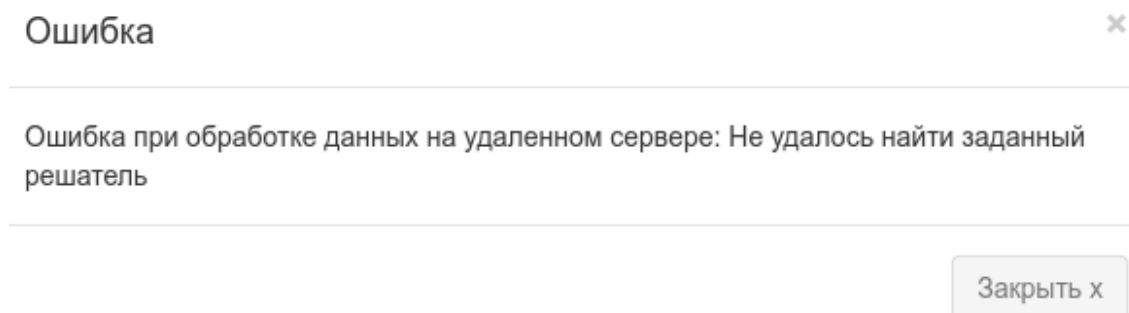


Рисунок 9 — Всплывающее окно с типом события и текстом, поясняющим произошедшее событие

В панели уведомлений был инкрементирован счетчик уведомлений. При раскрытии панели можно увидеть пришедшее уведомление о событии с иконкой обозначающей, что данное событие имеет тип «ошибка». Результат продемонстрирован на рисунке 10.

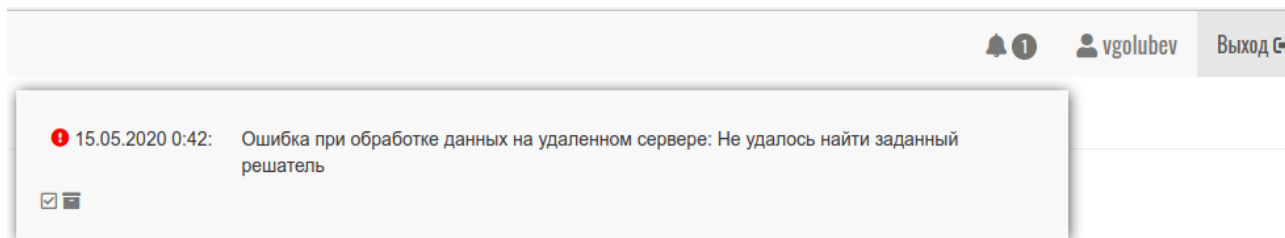


Рисунок 10 – Панель уведомлений в веб-клиенте, содержащая новые события

Для панели оповещений были разработаны функции для управления оповещениями. Функция «Отметить все оповещения как прочитанные» обнуляет счетчик оповещений у пользователя. Функция «Удалить оповещения» очищает панель уведомлений и удаляет оповещения, связанные с пользователями из временного хранилища веб-клиента.

4. АНАЛИЗ РЕЗУЛЬТАТОВ

Тестирование показало, что разработка полностью удовлетворяет заявленным требованиям. Новый «функционал» дополняет графический пользовательский интерфейс веб-клиента и помогает при решении задач в системе.

Для получения полного эффекта от работы подсистемы необходимо полностью интегрировать её во все существующие функции системы и применять предлагаемые интерфейсы при разработке новых функций. Возможным улучшением может оказаться интеграция встроенных инструментов Django для логирования событий в подсистему журнала системных событий. Для удобства использования со стороны пользователей системы возможна модификация пользовательского интерфейса и добавления новых функций в подсистемы возможности выбора подписки на различные типы уведомлений.

ЗАКЛЮЧЕНИЕ

В результате работы над подсистемой сообщений о системных событиях удалось добиться готового решения для внедрения в РВС GCD. Была разработана программная архитектура, связанная с существующим «функционалом» системы, значительно улучшен подход к обработке ошибок в системе. В пояснительной записке были описаны механизмы работы нового сервера приложений русомарс и протокол коммуникации comwps и русомарс, а также приведены рекомендации для возможного улучшения подсистемы журнала сообщений о системных событиях.

СПИСОК ЛИТЕРАТУРЫ

1. Винничек Е. В. Разработка программного комплекса централизованного логирования сложных распределенных систем // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и Технические Науки. – 2019. – №04. – С. 42-49.
2. Event Types // docs.microsoft.com: документация к Microsoft Windows. 2018. URL: <https://docs.microsoft.com/ru-ru/windows/win32/eventlog/event-types> (дата обращения: 13.05.2020).
3. Системный вызов syslog // opennet.ru: новости открытого ПО URL: <https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=syslog&category=3> (дата обращения: 13.05.2020).
4. Федченко И. О., Намиот Д. Е. Модель информационной системы на базе push-уведомлений // International Journal of Open Information Technologies. – 2015. – №8. – С.17-27.
5. Павлов А. Д., Намиот Д. Е. Системы для поддержки push-уведомлений //International Journal of Open Information Technologies. – 2014. – Т. 2. – №7. – С. 37-44.
6. Notification // developer.mozilla.org: Документация к Web API от Mozilla. URL: <https://developer.mozilla.org/en-US/docs/Web/API/notification> (дата обращения: 13.05.2020).

7. Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов. Программирование. – Т.47, №5 – 2019, с. 43-55.

8. M. Herlih. A Methodology for Implementing Highly Concurrent Data Object //

A

C

M

T

r

a

n

s

a

c

t

i

o

n

s

o

n

P

r

o

g

r

a

m

m

i